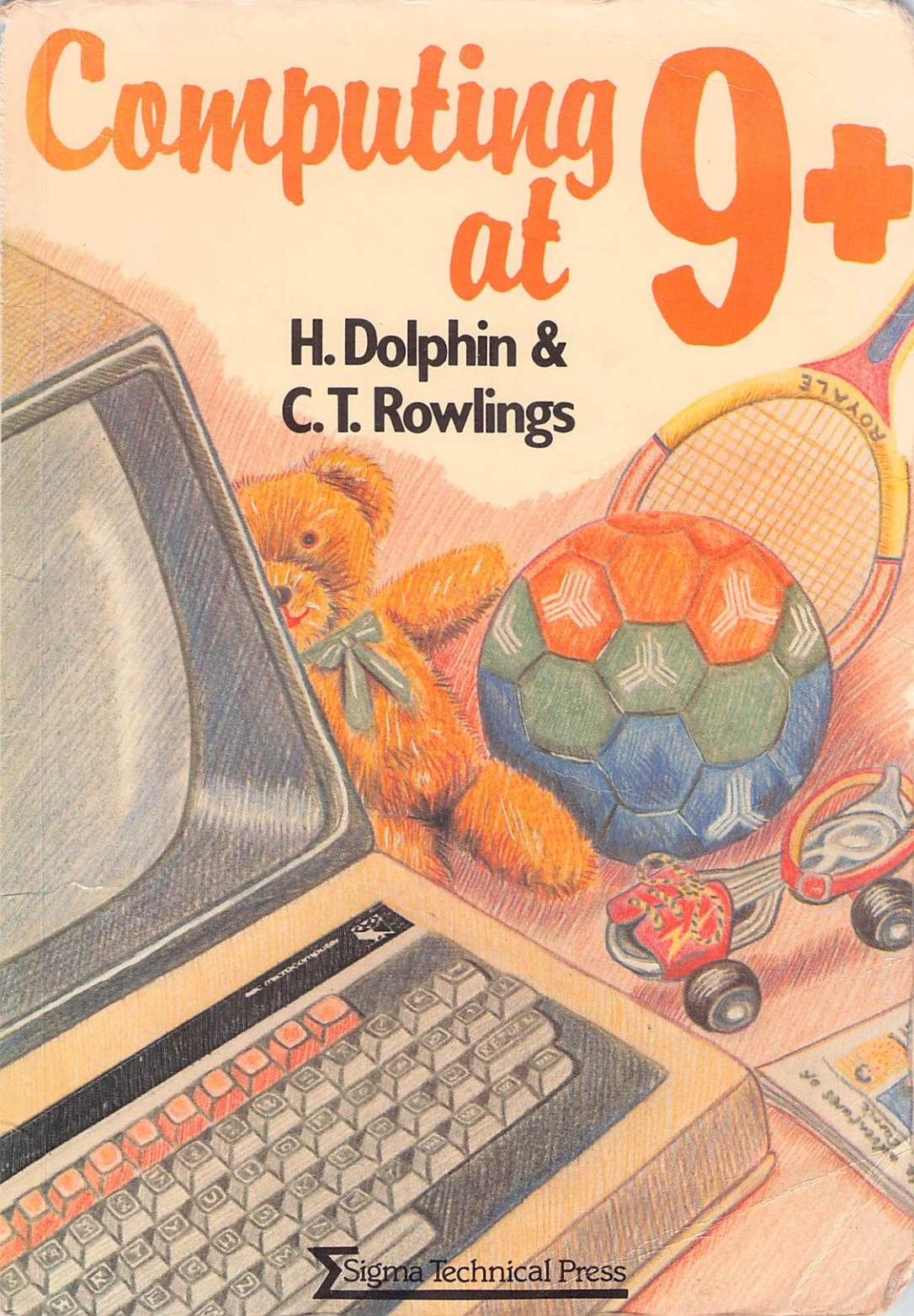
Computing at 9+

H. Dolphin &
C. T. Rowlings



Sigma Technical Press

COMPUTING AT 9+

by

H. Dolphin and C.J. Rowling

Sigma Technical Press

Copyright ©1984, by H. Dolphin and C.J. Rowling

All Rights Reserved

No part of this book may be reproduced by any means without the prior permission of the copyright owner. The only exceptions are for the purposes of review or for entering the programs herein onto a computer by the original purchaser of this book.

ISBN 0 905104 63 3

Published by:
SIGMA TECHNICAL PRESS
5, Alton Road,
Wilsmlsow,
Cheshire.
SK9 5DY.

Typesetting and Production by:
DESIGNED PUBLICATIONS LTD.
8-10 Trafford Road,
Alderley Edge,
Cheshire.
SK9 7NT

Distributed by:
JOHN WILEY AND SONS LTD.,
Baffins Lane,
Chichester,
Sussex,
PO19 1UD.

Printed and bound in Great Britain by
J. W. Arrowsmith Ltd., Bristol

CONTENTS

Introduction ix

Program Index vi

Chapter 1
How Your Computer Works 1
The Processor 1
How The Processor Works 2
Some Examples of How a Line is Interpreted by the Individual Micros 3
The ZX Spectrum 3
BBC and Electron 4
VIC 20 5
The Memory 5
Changing the contents of memory 6

Chapter 2
Programming a Microcomputer 7
Programs using LET 9
Changing the Program 10
Programs Using INPUT 11
Changing INPUT Programs 12
Programs Using LET and INPUT 13
Programs Using FOR and NEXT 13
Programs Using PRINT 14
Programs Using INPUT and $ 16
Programs Using GO TO 18
Programs Using the TAB Function 18
Conversion Programs 19
Programs Using READ and DATA 20
A READ and DATA Quiz Program 21
Re-Using READ and DATA 21
Comparing Programs 22
Programs Using GOSUB 23
Writing Your Own Programs 26
Programs Using STRINGS 29
Programs Using LEN 31
Programs Using DIM 32
Programs Using GET$ (INKEY$ for Spectrum owners) 35
Programs Using DEF FN 36
When your programs (gasp) DON'T WORK? Read on ... 37
Computer indicated errors 37
Design Errors 38

STRUCTURED PROGRAMMING - a way to make your programs work first time 39

Chapter 3

Some Mathematical Programs 42
EASY ARITHMETIC and SIMPLE PUZZLES 42
Guess the Number 43
Factors 45
Prime Numbers 45
Mathematical Patterns 46
Squares and Circles 47
HISTOGRAMS 48
SIMPLE ALGEBRA 50
SIMULTANEOUS EQUATIONS 51
FERMAT's Last Theorem 52
Quadratic equations 53
Simple trigonometry 54
"CHANGEGIVER" - a program design example 54

Chapter 4

Graphics 58
The Spectrum 58
BRIGHT 62
FLASH 62
INVERSE 63
User Defined Graphics 67
Graphics on The BBC 70
MODE, MOVE and DRAW 70
PLOT 73
Drawing Circles 75
Teletext Graphics 79
VIC 20 Graphics 82
Keyboard Graphics 82
Border Screen Colours 85
Graphics using CHR$ function 85

Chapter 5

Sound 89
Sound on the Spectrum 89
Spectrum Keyboard 91
Sound on the BBC computers 92
BBC Keyboard 95
Sound on the VIC 20 96
VIC Keyboard 97

PROGRAM INDEX

Aircraft (Spectrum) 67
Algebraic Equations 50
Anagram 122
Art? (BBC) 73
(Spectrum) 61

Battleships 33
Border (VIC) 85
Bubblesort 99

Car (BBC) 80
Cards (VIC) 86
Changegiver 55
Circle (BBC) 75
(Spectrum) 61
Circle area 20
Circle/Sphere 47
Coconuts problem 135
Co-ordinate guess 113
Counting to a million 134
Crash/bang (Spectrum) 69
Currency, foreign 130
Curve stitching (BBC) 71
(Spectrum) 59,60

3-D Patterns (Spectrum) 64
DEF FN 36
Depth Charge 113
DRAW (BBC) 70
(Spectrum) 59

Etch-A-Sketch (BBC) 74
(Spectrum) 66

Face (Spectrum) 66
Factoring 45
Female Animal 21
Fermaat's Theorem 52
FILE HANDLING (BBC) 104

Chapter 6
Sorting and Filing Information 99
Bubblesort 99
Insertion sort 100
Shell sort 101
Handling files on the BBC and VIC 102
BEEBFILER 104
VICFILER 105
File handling on the Spectrum 107

Chapter 7
Machine Code 109

Chapter 8
Fun and Games 112
THE 3-D GUESSING GAME 112
COORDINATE GUESS 114
Simple Logo 115
SCHOOL ADVENTURE 119
FURTHER SUGGESTIONS 122
ANAGRAM 123
SPEAD READ 124

Chapter 9
YOUR TURN NOW! 126
Projects 127
Suggested Answers 128

Chapter 10
THE FUTURE 137
The BBC machine and the Future 139
Printers and Disk Drives 139
Teletext adaptor 140
Prestel adaptor 140
The 1MHz Bus 140
Spectrum 141
Printers and Disk Drives 141
Other "add-ons" 142
VIC 20 143
Printers and Disk Drives 143
Other "add-ons" 143
Conclusion 144

Index 145

(Spectrum) 107
(Vic) 105
FOR/NEXT 13
Fortune Teller 131
Fraction Pie (BBC) 76
Funny Sentences 25,29

GET 36

Histogram 48

INKEY$ (Spectrum) 35
INPUT 11
INPUT$ 18
Insertion sort 100

Jumping Jack (BBC) 95
(Spectrum) 91
(VIC) 87

Keyboard (BBC) 95
(Spectrum) 91
(VIC) 97

LET 9
Letter counting 30
LOGO (BBC) 116
(Spectrum) 117
(VIC) 115
Lorry (VIC) 86

Machine Code 109
Miles/Kilometres 20
Morse Code 128
Multiplication 23,44

Name/Address 35
Number Guess 43,132
Number Patterns 46,129
Number Puzzle 127,134

Patterns (BBC) 71
(Spectrum) 61
Penny a day 128
PLOT (BBC) 73
(Spectrum)* 60
POKE/PEEK 7
PRINT with $ 14
Prime Numbers 45

Quadratic Equations 53

Rectangle (Spectrum) 64
Rule (VIC) 83

Ruler (VIC) 83

School Adventure 118
Sea Sounds (BBC) 94
Shell Sort 101
Simultaneous Equations 51
Siren (BBC) 93
(Spectrum) 89
(VIC) 97
Sound Envelope (BBC) 93
Space Invader (Spectrum) 68
Speed read 123
Square (Spectrum) 61
Square/Cube 47
Square Numbers 43
String Patterns (BBC) 71
(Spectrum) 60
Strings 29

TAB 19
Table square 32
Telephone puzzle 133
Teletext (BBC) 79,80
Test (BBC) 4
(Spectrum) 3
(VIC) 5
Triangles (BBC) 74
Trigonometry 54
Tunes (Spectrum) 90
Type a number 24

UDG (BBC) 76
(Spectrum) 67
(Vic) 82

Vicky (VIC) 84
Vowels 22

INTRODUCTION

This book is a programming course, a reference book and an ideas book all rolled into one. Although aimed mainly at the 9-16 age group, it suits all first-timers to computing.

Three models are considered in some detail although the Microsoft version of BASIC dealt with here will run on nearly all microcomputers. The three models are:

BBC Model A and B
Sinclair Spectrum
Commodore VIC 20

BBC programs also compatible with the Electron (except for the section on Teletext graphics and parts of sound). VIC 20 programs in this book are compatible with the Commodore 64, except for the size of the screen image area.

They all use the BASIC computer language, which was developed at Dartmouth College in the USA. The initials BASIC stand for Beginner's All-Purpose Symbolic Instructional Code. It was designed with commonly used English words so that beginners would not be frightened away from a computer.

We will be using a special version of the BASIC language, developed by Microsoft, an American software company. It is therefore called Microsoft BASIC. Many of the versions of BASIC now in use, for example Commodore BASIC and Sinclair BASIC, are very like Microsoft, but introduce new words to make the best possible use of the computer hardware.

CHAPTER 1

How Your Computer Works

Before describing how to program your microcomputer, it might be as well to have an overview of a computer system to see what happens when it is set to work.

Most microcomputers have a similar internal construction, as shown in Figure 1.1.

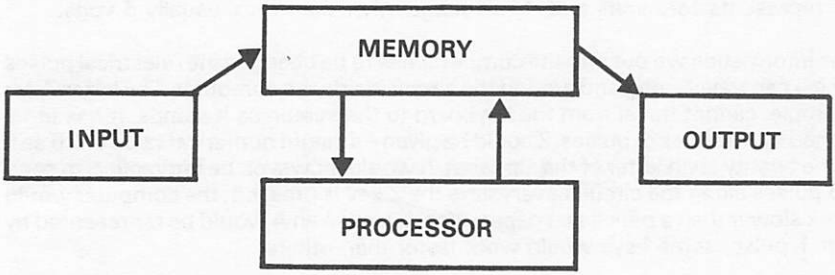


Figure 1.1: Computer Construction

The processor and memory are made of fingernail-sized pieces of silicon wafers known as integrated circuits or 'chips'. The input mechanism is usually a typewriter-like keyboard with some extra keys. The output device may be a TV screen or a printer.

Let's look in more detail at the processor and the memory.

The Processor

The processor, often called the Central Processing Unit or C.P.U. controls the whole operation of the computer system. It contains two important parts:

(a) a compiler or interpreter which automatically translates our BASIC program into a machine code language which can be executed by the

machine. The compiler has its own program to do this. Normally this special program cannot be altered and so it is stored in a Read Only Memory or ROM - more about this later.

(b) an arithmetic unit, which carries out the four mathematical operations: +, –, x, ÷ . There is also a "program counter" which ensures that these operations are carried out and that they don't get out of step with each other.

How The Processor Works

When we begin to type data into the computer, it is passed to the Central Processing Unit. Let us look at what happens.

The microcomputer is electrically powered and the individual parts of its circuits can either be 'on' or 'off'. These states can be represented by the binary digits '1' for ON and '0' for OFF.

'0' represents zero volts and '1' a voltage other than zero, usually 5 volts.

The information we put into the computer has to be changed into electrical pulses which can travel along and around the circuit inside the computer. The letter Z, for example, cannot travel from the keyboard to the screen as it stands. It has to be turned into a series of pulses. Z could be given a straight numerical value of 26 as it is the twenty sixth letter of the alphabet. It would, however, be impractical to send 26 pulses along the circuits every time the Z key is pressed: the computer would work slower than a pencil and paper. Also, because an A would be represented by just 1 pulse, some keys would work faster than others!

Instead, the computer represents each letter as a hexadecimal (or hex) number, i.e. a number in a system working to a base of 16. We can compare denary (base 10) and hexadecimal (base 16).

Denary (base 10)	1	5	10	15	20	25	30	35
Hexadecimal (base 16)	01	05	0A	0F	14	19	1E	23

The computer keeps a large table of hex numbers: one for each character that it has on the keyboard or that can be displayed on the screen.

A further conversion is made from hexadecimal to binary code (working to base 2), before the computer can interpret which ke was pressed. The letter Z, for example, has the value 00011010 in the binary code, made up like this:

128	64	32	16	8	4	2	1
0	0	0	1	1	0	1	0

The letter Z has now been converted into eight pulses or 'bits' which can be sent along the circuits of the computer. This pattern of 8 bits is called a "byte".

To see how *YOUR* computer actually translates what you type on its keyboard, you have to find a way of looking inside its memory. Each of the microcomputers we are covering has a somewhat different memory layout, so we will put the one-line program:

```
10 PRINT "TEST"
```

into each machine and then find a way of looking at what has been stored in memory.

Even though each computer uses a different processor chip (6502 on the BBC, Electron and VIC-20, 6510 on the Commodore 64 and Z80 on the Spectrum) the way that your program is stored is very similar. Let's look at some examples:

SOME EXAMPLES OF HOW A LINE IS INTERPRETED BY THE INDIVIDUAL MICROS

The ZX Spectrum

A useful tip for Spectrum owners is, at the start of any programming, put in POKE 23609,100. This will give a 'click' sound every time a keystroke is executed. So, type in POKE 23609,100 and press ENTER.

Now type in this program:

```
10 PRINT "test"
```

When we press ENTER, the line of print will move from the bottom to the top of the screen. The computer remembers this short program in its memory. Do not worry about what we are about to do; it looks very complicated but all it needs is care in typing in. Type in this next line but do not press ENTER until you have typed it in completely:

```
FOR t = 23755 TO 23765 : PRINT PEEK t;" ";:NEXT t
```

Now press ENTER.

Your program 10 PRINT "test" will disappear, to be replaced by this line of numbers at the top of the screen:

```
0 10 8 0 45 34 116 101 115 116 34
```

Now turn to pages 183 to 188 of the Sinclair ZX Spectrum BASIC programming manual. Do not worry about the column headings, as they will be explained in the chapter on Machine Code. Look at the first column of numbers headed "Code". Find the number 245 on page 188 and you will see that it represents the command PRINT.

Number 34 represents the quotation marks ". Can you find the code for the letters in the word 'TEST'?

Type in the number 201 to return the computer to BASIC language.

We can, of course, look at any program in the same way. The only thing we need to check is that the range of numbers in the FOR section (which is the range of memory being examined) is big enough to show all the numbers that the computer has changed our program into.

We can look at the characters that the Spectrum can print - the Character Set - using the program:

```
10 FOR C = 33 TO 255 <- press ENTER

20 PRINT C, CHR$C <- press ENTER

30 NEXT C <- press ENTER
```

The scroll at the bottom of the screen allows you to keep on looking at the characters until 255 is reached. You do this by pressing Y for 'Yes'.

BBC and Electron

First, type in this program:

```
10 PRINT "TEST"
```

RUN the program and then type the following, pressing RETURN at the end:

```
FOR I%=PAGE TO TOP:PRINT;?I%;" ";:NEXT I%
```

You will now be able to see a display which looks like this:

```
13 0 10 11 241 34 84 69 83 84 34 13 255 32
```

This is the way the computer has stored your program. By using your BBC User Guide you can now refer to the section at the back which shows the "Alphanumeric Characters" (page 486). Here you will find the codes for each of the letters used in the program. For example, 34 is the code for quotation marks. Can you find the codes for the letters in the word TEST? They should match what you can see on the screen. You could look at any program in this way and see the whole of the program scroll before you!

To see all the characters your machine can print, and their values, try typing in:

```
FOR I=32 TO 255:PRINT;I,CHR$(I):NEXTI
```

To stop the display whizzing past your eyes, hold down the SHIFT and CTRL keys together.

VIC-20

First, type in this program:

```
10 PRINT"TEST"
```

RUN the program and then type the following, pressing RETURN at the end:

```
FOR I=4096 TO 4108:PRINT PEEK(I);" ";:NEXT I
```

You will now be able to see a display which looks this:

0 13 16 10 0 153 34 84 69 83 84 34 0 0

This is the way the computer has stored your program. By using your VIC User Guide you can now refer to the section which shows the Character Set. Here you will find the codes for each of the letters used in the program. For example, 34 is the code for quotation marks. Can you find the codes for the word TEST? They should match what you can see on the screen. You could look at any program in this way and altering 4108 to a higher number, you can see the whole of the program scroll before you!

To see all the characters your machine can print and their values, try typing in:

```
FOR I=32 TO 255:PRINT I,CHR$(I):NEXTI
```

To slow down the display, press the CTRL key.

The Memory

Your computer has two sorts of memory:

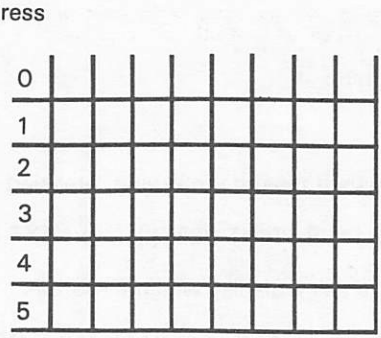
Read Only Memory (ROM) and Random Access Memory (RAM)

ROM stores permanent things like the BASIC interpreter.

In contrast, the Random Access Memory must be programmed afresh each time a program is typed in. The RAM was used to store the "TEST" program we just looked at.

RAM can be imagined as a very large number of pigeon holes. Each of the compartments is given a permanent identifying label or 'address'. In each compartment, 8 single bits (equal to one 'byte) of information can be stored:

Each cell or pigeon hole can contain a single binary digit either '1' or '0'. Each group of cells is a 'byte'. Each byte represents a single number or letter. When information is put into the RAM, it overwrites any information previously stored at that address. But we can keep on recalling the information at an address without altering it.



The size of the memory is measured in 'Ks'. 1K represents 1024 bytes or bits of information. A microcomputer with a 16K memory can store 16 x 1024 bytes of information less the bytes needed by the computer's own operating system.

Changing the contents of memory

We can put information at a memory address and later see what is stored there. The POKE statement causes information to be stored at almost any memory address we require. The opposite of POKE is PEEK. The PEEK statement allows us to look at the contents of a memory address without the contents being affected.

Let's use POKE and PEEK to see the system at work.

We will use POKE to print near the top centre of the screen.

Spectrum	**BBC**	**VIC 20**
POKE 16400,66 Press ENTER PRINT PEEK 16400 Prints 66 at the top of the screen	? 31760=66 PRINT ? 31760 ? means PEEK and POKE in BBC BASIC	POKE 7777,66 PRINT PEEK 7777

CHAPTER 2

Programming a Microcomputer

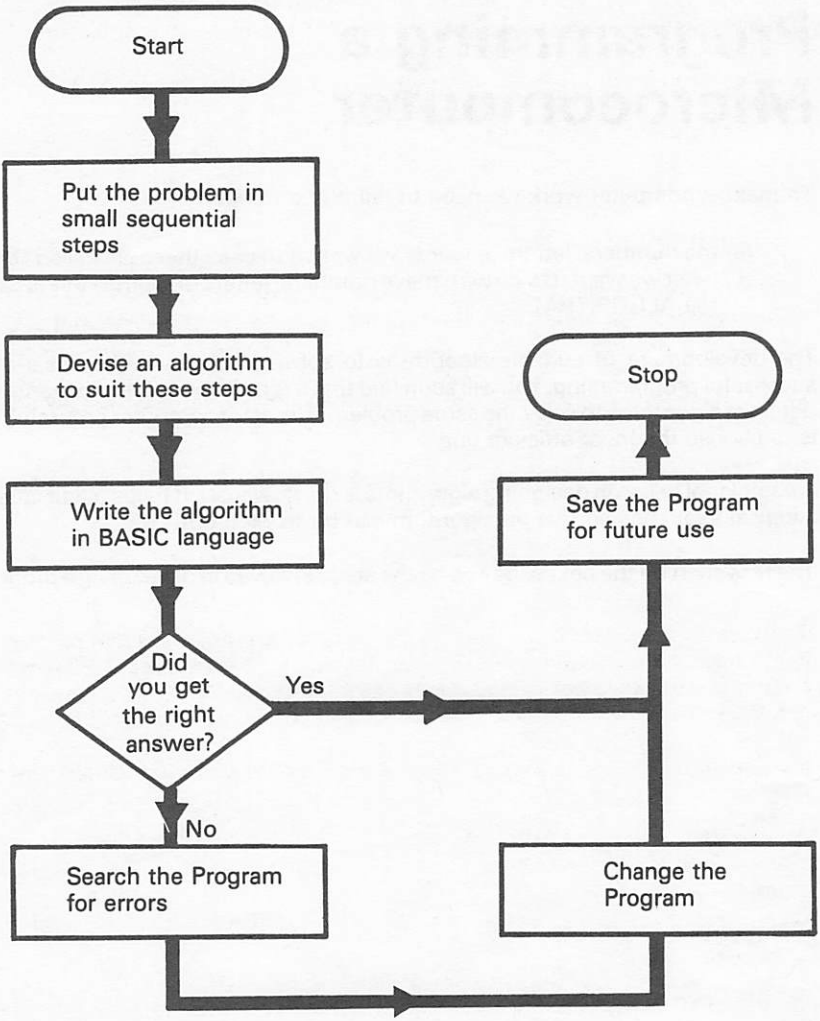
To make a computer work we need to tell it two things:

(a) the numbers, letters or words we want it to use - these are called DATA.
(b) what we want it to do with these numbers, letters or words - this is called the ALGORITHM.

The development of suitable algorithms to solve problems is the basis of all successful programming. You will soon find that it is quite possible to have several different algorithms to solve the same problem. The art of computer programming is to choose the most efficient one.

A useful tool to use in designing algorithms is the flowchart. It helps to put ideas in a logical sequence so that an algorithm can be worked out.

The flowchart on the next page shows the steps involved in developing a program.


Start
Put the problem in small sequential steps
Devise an algorithm to suit these steps
Write the algorithm in BASIC language
Did you get the right answer?
Yes
No
Search the Program for errors
Change the Program
Save the Program for future use
Stop

Later in this chapter we will use a flowchart to write an actual program.

We will be using BASIC to program the microcomputer. It was one of the first easy-to-use languages to be developed and attempts to use familiar words to identify particular processes or operations. The word NEW, for example, tells the computer that the program shortly to be typed in has nothing to do with any program already stored in the machine's RAM memory, so it erases the RAM.

The words in BASIC's vocabulary can be divided into:

(a) System commands

and

(b) BASIC keywords

System commands instruct the computer to carry out specific tasks. They are usually entered before or after the program is run so that the computer knows what to do with the program.

BASIC keywords are words or abbreviations used in the program so that the computer knows what to do with the DATA we have entered.

We will use both commands and keywords in the following programs: that follows:

Programs using LET

Suppose that we want to tell the computer something and have it remember this fact for some time. The word LET is used to do this. If we enter LET A = 4 the computer will remember the value of 4 in the variable called A. The value of A will remain at 4 until we tell the computer to change it.

If we want to add two numbers, say 4 and 3, the program would look like this:

```
NEW
10 LET A = 4
20 LET B = 3
30 PRINT A + B
40 END
RUN
```

NEW and RUN are system commands. NEW tells the computer to erase all information stored in its RAM - so always be careful when using NEW as it can destroy hours of work.

The command RUN instructs the computer to carry out the program we have entered.

After putting in any system command or line of BASIC program the RETURN or ENTER key is pressed to send the command or line to the CPU. Each line of the actual program is given a number so that the computer can keep them in the right order. We number these lines in 10s so that we can put new lines in between the old ones by giving them numbers like 15 or 25. (You can type in a line of BASIC without a line number but the computer will then perform the line immediately. The computer can be used like a calculator in this immediate mode: Numbered lines are only obeyed when we type RUN) try entering "PRINT 4+3" and pressing ENTER or RETURN.

Changing the Program

We can change a program already in the computer by using a LIST command to show us the program and then re-typing the line we wish to change. This line deletes the existing line when the RETURN key is pressed.

Suppose we want to make the LET program into one which does multiplication; line 30 tells the computer to add A and B, so it is this line which needs to be changed.

We LIST the program and type the new line underneath like this:

```
30 PRINT A * B
```

Now press RETURN or ENTER to put the new line into the program where it will take the place of the old line.

The Sinclair Spectrum has another way of changing lines which is useful when the line we want to change is complicated. To change the same line, the Spectrum owner types LIST 30 and when the computer has listed the program from line 30 onwards, CAPS SHIFT and EDIT are pressed together. Line 30 will be printed on its own at the bottom of the screen. Use the CAPS/SHIFT and cursor control keys (those with white arrows ← ↓ ↑ →) to move the cursor behind the sign, letter or number to be changed. When the cursor is behind the offending character, press CAPS/SHIFT and DELETE together, which will remove the unwanted character. Simply key in the new sign, letter or number in its correct place and press ENTER to put the changed line back into the program.

BBC machines also have another method of changing lines which is useful when the lines are long or complicated. To change line 30, follow this procedure:

Type LIST 30
Press the up cursor key ↑ until the cursor is on the same line as the listed line 30.
Now press the COPY key one press at a time until the cursor is on the +.
As you do this you will see the line being copied out underneath.
Now press * and move the cursor past the old + by one place using the → key.

Press the COPY key again until the line has all been copied.
Press RETURN and the new line is now in the computer's memory.

Type LIST to check the line is correct.

Commodore machines use cursor keys to perform "on-screen editing".

To change line 30 follow this procedure:

TYPE LIST 30.
Use the cursor keys to position the cursor right on top of the character to be changed.
Type *
Press RETURN.
Check that the line has been changed by typing LIST 30 again.

Until now we have used the letters A and B to help the computer to remember numbers. We can use any other letters to do this. Try using P and Q instead of A and B like this:

```
NEW
10 LET P = 8
20 LET Q = 4
30 PRINT P + Q
40 END
RUN
```

Programs Using INPUT

A more useful group of programs uses the word INPUT which tells the computer that we want to enter some letters, numbers or words while the program is running. We have used different letters, X and Y, for the numbers we will enter when the program is running.

Type in this program:

```
NEW
10 INPUT X,Y
20 PRINT X + Y
30 END
RUN
```

When we type RUN, the BBC (and VIC 20) will print a ? which means that the computer is ready for us to type in the numbers which we want to add together. The ZX Spectrum asks for these numbers with an L at the bottom of the screen. Type in two numbers separated by a comma. Press RETURN or ENTER and the computer will give the answer to the sum we have asked it to work out at line 20.

The INPUT program is very useful because we can use the same program again and again to add different numbers together without re-typing any lines.

The INPUT program is used again by typing in the command RUN and pressing RETURN or ENTER. The computer will show with the ? or L that it is ready for two new numbers to be typed in.

We can stop the program running by pressing the RUN STOP, BREAK or ESCAPE key on the computer.

Changing INPUT Programs

Here are some slightly different programs.

The first program subtracts the number represented by G from the number represented by F.

```
NEW
10 INPUT F,G
20 PRINT F - G
30 END
RUN
```

By re-typing line 20 and using the LIST command, change the program to one which will multiply the numbers we type in:

```
10 INPUT F,G
20 PRINT F*G
30 END
RUN
```

We use the letter Z in the next program to make the computer remember the third number we put in. Line 20 tells the computer to multiply these three numbers. By re-typing lines 10 and 20 we can change the program to:

```
10 INPUT F,G,Z
20 PRINT F*G*Z
30 END
RUN
```

Now let's see if you can change this to:

```
10 INPUT F,G,Z
20 PRINT F+G+Z
30 END
RUN
```

This program will add together the three numbers we entered.

Programs Using LET and INPUT

We can use the words LET and INPUT in the same program. Suppose we want to work out some answers in the seven times table.

Line 10 tells the computer to keep one number as 7. Line 20 tells the computer that we are going to put in four numbers. Line 30 tells the computer to multiply each of these numbers by 7.

```
NEW
10 LET T = 7
20 INPUT A,B,C,D
30 PRINT A*T,B*T,C*T,D*T
40 END
RUN
```

We can keep on re-using this program by typing RUN and pressing ENTER when we have looked at the answers to the first four sums. The computer shows with an ? or L that it is ready for another four numbers to be entered. Again each one will be multiplied by 7 when we type RUN and press ENTER.

Can you change this program to work out four different multiples of 10?

Here's some help if you are stuck. The line which tells the computer that we want to work with a particular number is 10 and so this is the line which needs to be changed. So, just type:

```
10 LET T = 10
```

Programs Using FOR and NEXT

We now move to a new sort of program which uses the new words FOR and NEXT. These are always used together.

```
NEW
10 FOR N = 1 TO 10
20 PRINT N,2*N
30 NEXT N
RUN
```

FOR tells the computer to take the first number, 1, and use it in line 20. So the computer would print 1 2.

The word NEXT in line 30 tells the computer to take the next value of N in the sequence 1 to 10 and then use it in line 20. The computer will keep on doing this until N is greater than 10, when the program will stop.

RUN the program and see the great speed and accuracy of the computer. It prints out the answers to the two times table in less than a second.

How would we change this program to make the computer look at the numbers from 1 to 20?

Line 10 needs to be changed to read:

```
10 FOR N = 1 TO 20
```

How can we make the computer print 3 times N?

```
20 PRINT N,3*N
```

We can put FOR and NEXT loops inside each other. This is called "nesting". When we do this we must be sure that the inner loop starts and finishes inside the outer loop.

Try this program - we have inset the innner loop so that it is easy to check that it both starts and finishes inside the outside loop.

```
10 FOR N=1 TO 3
20     FOR I = 1 TO 2
30         PRINT I
40     NEXT I
50 NEXT N
```

This program prints out three lots of I because N in line 10 can be 1, 2 or 3.

The task given to the computer in lines 20, 30 and 40 is then carried out three times.

Try altering the range given to N in line 10. What difference does your alteration make ?

Now try altering the range given to I in line 20. Does the program do what you expected ?

Spend a fair amount of time experimenting with these routines because they are very important parts of some programs given later in the book.

Programs Using PRINT

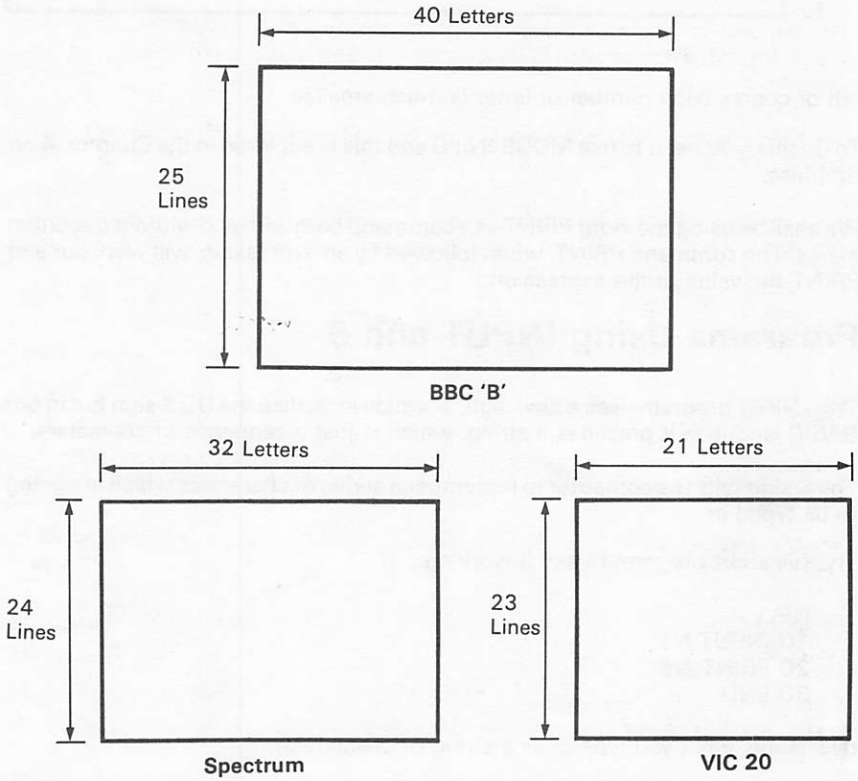
We have been using the word PRINT to make the computer work out and print the answers to sums. We can also use the word PRINT with quotation marks to print exactly what we put inside quotation marks - it will copy any spaces, mistakes etc which we may carelessly put inside these quotation marks.

Try this program, entering your own and your friend's name:

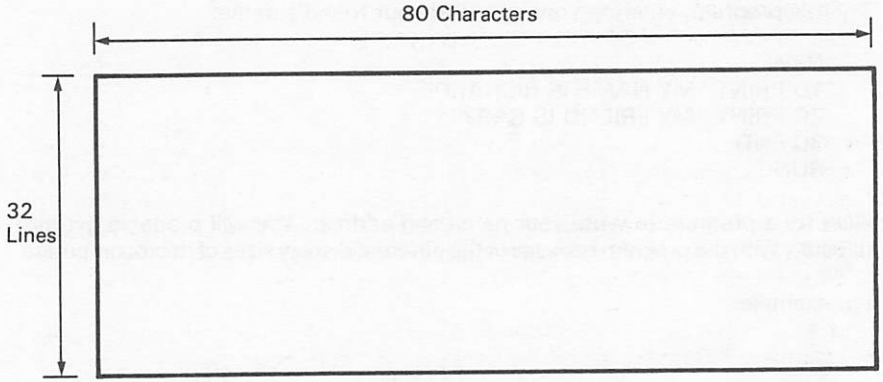
```
NEW
10 PRINT "MY NAME IS RICHARD"
20 PRINT "MY FRIEND IS GARY"
30 END
RUN
```

Now try a program to write your name and address. We will probably get into difficulty with the program because of the different display sizes of microcomputers.

For example:



We have to rearrange our program so that it will fit the display we have. BBC model B microcomputer users are fortunate because they can increase the size of their display to:



but of course each number or letter is much smaller.

To do this, you need to use MODE 3 or 0 and this is explored in the Chapter 4 on Graphics.

We shall be using the word PRINT as a command both with and without quotation marks. The command PRINT, when followed by an expression, will work out and PRINT the value of the expression.

Programs Using INPUT and $

This INPUT program uses a new sign, $, which looks like the US $ sign but in our BASIC language it precedes a string, which is just a sequence of characters.

The $ sign tells the computer to remember a series of characters which are going to be typed in.

Try this short program to see it working:

```
NEW
10 INPUT N$
20 PRINT N$
30 END
```

(N$ stores what you type in as a string of characters)

Now let us look at a more complicated program:

```
NEW
10 PRINT "WHAT IS YOUR NAME? "
20 INPUT N$
30 PRINT "WHO IS YOUR FRIEND? "
```

```
40 INPUT F$
50 PRINT "SO";N$;"YOUR FRIEND IS ";F$
60 END
RUN
```

Line 10 makes the computer print 'What is your name?' We put in our name by using the question mark (or at the L sign on the Spectrum). The computer will remember our name as N$ and be able to use it in line 50. The $ sign in line 40 makes sure the computer remembers our friend's name as F$ when it has been put in. So the computer is now remembering two names, one that it calls N$ and the other it calls F$. It uses both these names in line 50.

The display will be hard to read because the words are so close together. We need to experiment to find the best layout for our particular computer.

Try changing line 50 to:

```
50 PRINT "SO",N$,"YOUR FRIEND IS",F$
```

If this display is not attractive try another line 50, being *VERY CAREFUL* with the spaces:

```
50 PRINT "SO ";N$;" YOUR FRIEND IS ";F$
```

As you can see, a comma puts more space between each item, whereas a semi-colon places them close together.

We can write a slightly more complicated program in a similar way:

```
NEW
10 INPUT "WHAT IS YOUR NAME? "; N$
20 PRINT "NOW ";N$;" WHICH IS YOUR CLASS?"
30 INPUT C$
40 INPUT "WHICH IS YOUR SCHOOL? " ; S$
50 PRINT "SO ";N$;" YOU ARE IN ";C$;"AT ";S$
60 END
```

Lines 10 and 40 show a new way of using INPUT. In the previous program we used a PRINT line to ask a question and an INPUT to accept the answer. In this program the word INPUT is being used with quotation marks, asking for the answer at the end of the line.

We use the word INPUT with quotation marks again in line 40.

Line 50 prints a sentence which uses the answers we have given in lines 10, 30 and 40.

Now change the program so that lines 20 and 30 are combined into one INPUT line.

Programs Using GO TO

When the computer reaches a line with the command GO TO it will immediately move to the line given in the GO TO. There is great danger when using the GO TO because the computer jump may over important lines in the program. The program will then be short of instructions or information, so be careful!

In this program, we have used the PRINT with quotation marks to make a pattern around your name with the multiplication sign. You will probably have to experiment with the length of pattern on lines 10 and 30 and with the spaces on line 20 to space out your own name correctly.

```
NEW
10 PRINT "* * * * * * * *
20 PRINT "*  J A N E *
30 PRINT "* * * * * * * *
40 GO TO 10
END
```

Make sure that you do not mix up O the letter, and zero the number.

The GO TO on line 40 will make the computer jump to line 10 and begin again. The computer will keep on printing your name until you tell it to stop by using the RUN/STOP, BREAK or ESCAPE key.

The display can be improved by putting empty lines of spaces above and below the name like this:

```
11 PRINT "*              *"
31 PRINT "*              *"
```

Now try this program using the same idea but with INPUT and $:

```
NEW
10 INPUT "PLEASE PUT IN YOUR NAME ";N$
20 PRINT "* * * * * * * *"
30 PRINT "*",N$,"*"
40 PRINT "* * * * * * * *
50 GO TO 20
```

Again we will have to experiment with the layout of the display until it is as we want it to be.

Programs Using the TAB Function

The word TAB moves the beginning of PRINTing across the screen. The number

tells the computer how many spaces it has to move across the screen. TAB (10) moves the PRINTing 10 spaces across the screen.

This program prints a name 20 times, so that the names slope from left to right:

```
10 FOR M = 1 TO 20
20 PRINT TAB (M);"LOUISE"
30 NEXT M
```

Now let us make the names slope the other way, from right to left:

```
10 FOR N = 20 TO 1 STEP-1
20 PRINT TAB (N); "ANN"
30 NEXT N
```

STEP-1 asks the computer to count backwards from 20 to 1.

The programs can be combined to give an attractive display:

```
10 FOR M = 1 TO 5
20 PRINT TAB (M);"LOUISE"
30 NEXT M
40 FOR N = 5 TO 1 STEP-1
50 PRINT TAB (N);"ANN"
60 NEXT N
```

Conversion Programs

We can use our knowledge of BASIC to write a program which is really useful because it does lots of complicated sums for us. It prints two different measures of distance - miles and kilometres. One measure - miles - is from the old Imperial system used in the United Kingdom until metrication replaced it. The kilometre is from the Metric system.

Changing from one to the other needs care. Let us write a program to do this. We know that a kilometre is 5/8 of one mile. This gives us line 20, in the program below, in which each kilometre is worked out in parts of a mile.

Lines 10 and 50 tell the computer to take every mile from 1 to 20. Line 40 tells the computer to print Miles and Kilometres side by side under "MILES" "KILOMETRES" from line 30.

After using - and understanding - this program, can you write a program to change pints to litres? A litre = 1.76 pints.

```
NEW
10 FOR M = 1 TO 20
20 LET K = M/(5/8)
30 PRINT "MILES","KILOMETRES"
40 PRINT M,K
50 NEXT M
60 END
RUN
```

Programs Using READ and DATA

An important and useful group of programs use the words READ and DATA. The word READ tells the computer to look for a line in the program beginning with the word DATA, and to take from it the first number, letter or word to be used in the program.

The word DATA tells the computer that a line of words, numbers or letters is not to be used until the instruction READ tells the computer to use the DATA line.

In the following program we have used the command GO TO in line 50 so that the computer, having picked the first number from the DATA line and carried out the instructions in lines 20 and 40, can pick the second number from the DATA line and carry on.

The DATA line can be put almost anywhere, but it must not be inside the return loop made by line 40, otherwise the program will run out of DATA and print an error message.

```
NEW
10 READ R
20 LET A = 3.14159*R*R
30 PRINT "RADIUS","AREA"
40 PRINT R,A
50 GO TO 10
60 DATA 1,2,3,4,5,6,7,8,9,10
70 END
RUN
```

This program will work out the area of circles - the radius of each circle is in the DATA line 60. When we RUN the program the computer will work out the area of the ten circles and then say OUT OF DATA ERROR LINE 10. Do not worry: this is the computer's way of telling you that it has used the DATA we gave it in line 60. Add line 70 - 70 DATA 11,12,13,14,15,16,17,18,19,20 and RUN again.

A READ and DATA Quiz Program

This program shows how we can make a Quiz program. We will ask for the names of the females of some animals. Let us look at the lines of the program.

The Quiz instructions are in lines 10,20 and 30.

Line 40 tells the computer to read the animal's name and its female form from the DATA lines.

Line 60 makes the computer show a ? where we put in the female form of the animal.

Line 70 checks if the female name we have just put in is the same as the correct answer in the DATA line. If it is, the computer goes to line 100 and prints its congratulations. If the answer does not match, it goes to line 80 which tells us of the mistake and then gives the right answer.

Line 90 tells the computer to go back to line 40 so that the next DATA line is used.

Lines 120 to 160 are DATA lines where the name of the animal and its female form are shown together.

```
NEW
10 PRINT "I WILL GIVE YOU AN ANIMAL. CAN YOU"
20 PRINT "GIVE THE FEMALE? FOR EXAMPLE:"
30 PRINT "IF I PUT RABBIT YOU PUT DOE.
40 READ A$,F$
50 PRINT A$
60 INPUT G$
70 IF G$ = F$ THEN 100
80 PRINT "SORRY THE ANSWER WAS ";F$
90 GO TO 40
100 PRINT "WELL DONE. TRY ANOTHER.
110 GO TO 40
120 DATA DOG,BITCH
130 DATA FOWL,HEN
140 DATA DEER,DOE
150 DATA CATTLE,COW
160 DATA HORSE,MARE
170 END
RUN
```

Re-Using READ and DATA

Both these READ and DATA programs use the DATA line once only. We may want to use the DATA line again and again.

When we use this Quiz program we want to use the whole DATA line when the second answer is put in. The word RESTORE lets us do this. Type in this program:

```
10 INPUT "PUT IN A VOWEL";W$
20 RESTORE
30 READ V$
40 IF W$ = V$ THEN 90
50 IF V$ = "STOP" THEN 70
60 GO TO 30
70 PRINT "SORRY ";W$;" IS A CONSONANT"
80 GO TO 10
90 PRINT "WELL DONE. TRY ANOTHER"
100 GO TO 10
110 DATA A,E,I,O,U,STOP
120 END
```

Line 10 asks us to put in a vowel which the computer will remember as W$. Line 30 makes the computer search the DATA lines. It calls the answer that it finds V$. Each of these answers is compared to our input W$ in line 40. If we are right, the computer goes to line 90. If the computer reaches STOP in the DATA line without finding a match for W$ it is sent to line 70 where we are told that our letter is a consonant. The GO TO in line 60 makes the computer compare each letter in the DATA line until it reaches STOP. The other two GO TO commands in lines 80 and 100 make the computer go to line 10, where we are asked to 'Put in a vowel'.

Comparing Programs

Different programs can give the same output. The best ones are those which are shortest to write and quickest to use.

The following four programs work differently but will actually give the same answers. Each program has a line which asks the computer to print 1 x N and 2 x N.

Program 1

```
10 FOR N = 1 TO 20
20 PRINT 1*N,2*N
30 NEXT N
40 END
```

Program 2

```
10 INPUT N
20 PRINT 1*N,2*N
30 GO TO 10
END
```

Program 3

```
10 READ N
20 PRINT 1*N,2*N
30 DATA 1,2,3,4,5,6,7,8,9,10
40 DATA 11,12,13,14,15,16,17,18,19,20
50 GO TO 10
60 END
```

Program 4

```
10 LET N = 1
20 PRINT 1*N,2*N
30 IF N=20 THEN 60
40 LET N = N + 1
50 GO TO 20
60 END
```

The final program has two unusual lines 30 and 40. Line 30 sends the computer to line 60, which ends the program, when N becomes greater than 20. Line 40 adds 1 to N.

Which program do you think is the best?

The worst is probably the second, the INPUT program because we have to put in each number from 1 to 20.

The shortest and quickest program seems to be the first, the FOR and NEXT one. It takes less time to type in (except for the "INPUT" program) and is quickest in giving us the answers.

Programs Using GOSUB

The word GOSUB stands for GO SUBroutine. A subroutine is part of a program which we need to use again and again. Using GOSUB-RETURN saves us having to type in the same routine in different parts of the program. We can think of a subroutine as a small program inside a larger one.

When we set the program to work, the computer works through it line by line until it reaches GOSUB. The computer is then instructed to go to a particular line and works through the subroutine until it reaches a RETURN. This sends the computer back to the main program, rejoining it at the line after the GOSUB.

Try this simple program and you will see how the computer jumps to the subroutine to print out the answer over and over again. Notice that line 999 has a REM or REMark statement in it. This is to help us REMember what the subroutine is for. If we put anything after a REM statement, the computer just ignores it.

```
10 INPUT"TYPE A NUMBER";X
20 GOSUB 1000
30 GOTO 10
999 REM ******SUBROUTINE*******
```

```
1000 PRINT
1010 PRINTX;" MULTIPLIED BY 100 IS ";X*100
1020 PRINT
1030 RETURN
```

Try altering the 100s in line 1010 and make sure the program works with both decimals and negative numbers.

Here is a different sort of GOSUB:

Imagine that we need a program which will jump to one subroutine if X is 4, but to a different subroutine if X is 6. We can do this by using ON ...GOSUB.

The ON part is a sort of multiple IF statement. If X is 1 then the computer will jump to the first subroutine number listed after the word GOSUB. If X is 2 it will jump to the second subroutine after the word GOSUB and so on. You can see this working quite clearly in the program below. Line 30 is the controlling line. If you type in 1 then it jumps to 100 and back, if you type in 2, it jumps to 200 and back and so on.

```
5 PRINT
10 INPUT"TYPE A NUMBER FROM 1 TO 9";X
15 PRINT
20 IF X>9 OR X<1 THEN PRINT "1 - 9 ONLY";GOTO 5
25 PRINT"YOU TYPED ";
30 ON X GOSUB 100,200,300,400,500,600,700,800,900
40 GOTO 5
100 PRINT"ONE": RETURN
200 PRINT"TWO": RETURN
300 PRINT"THREE": RETURN
400 PRINT"FOUR": RETURN
500 PRINT"FIVE": RETURN
600 PRINT"SIX": RETURN
700 PRINT"SEVEN": RETURN
800 PRINT"EIGHT": RETURN
900 PRINT"NINE": RETURN
```

(Spectrum owners should replace line 30 by the lines 30 if x=1 gosub100, 31 if x=2 gosub200 etc)

We can use GOSUB to "structure" our program i.e. we can break the program down into smaller sub-programs and write each part instead of trying to write the whole thing at once. In the program that follows you can see that the program has been broken down into subroutines and the main loop just calls each subroutine in turn. These are labelled in the program with REMark statements. There may be lines that you do not yet understand fully, but type them in carefully and RUN the program. It will seem as if that the computer is writing its own sentences!

You could change each of the words in the DATA statements to your own, but make sure you replace nouns with nouns, adjectives with adjectives etc. or you'll get some really crazy sentences.

```
10 REM****MAIN PROGRAM LOOP****
20 GOSUB 80:REM DATA
30 GOSUB 150:REM RANDOM SELECTION
40 GOSUB 170:REM PRINT SENTENCE
50 GOSUB 210:REM TIME DELAY
60 GOTO30
70 REM****READ****
80 DIM AR$(2),AD$(10),NO$(10),VE$(10),AV$(10)
90 READ AR$(1),AR$(2)
100 FOR I = 1 TO 10
110 READ AD$(I),NO$(I),VE$(I),AV$(I)
120 NEXT I
130 RETURN
140 REM***RANDOM***
150 RI=RND(2):R2=RND(10):R3=RND(10):R4=RND(10):R5=RND(10)
160 RETURN
170 REM***PRINT***
180 PRINTAR$(R1);" ";AD$(R2);" ";NO$(R3);" ";VE$(R4);" ";AV$(R5);"."
190 RETURN
200 REM***DELAY****
210 FOR T=1 TO 1000:NEXT T
220 RETURN
230 REM****DATA****
240 DATA THE,A,BLACK,CAT,SAT,QUIETLY
250 DATA NAUGHTY,BOY,CRIED,LOUDLY
260 DATA LAZY,HORSE,SLEPT,PEACEFULLY
270 DATA GOOD,GIRL,WORKED,QUICKLY
280 DATA FIERCE,DOG,BARKED,NOISILY
290 DATA TALL,MAN, TALKED,DEEPLY
300 DATA HAPPY,BLACKBIRD,SANG,SOFTLY
310 DATA TINY,MOUSE,SQUEAKED,SWEETLY
320 DATA SLIMY,SNAIL,SLITHERED,SLOWLY
330 DATA SILLY,PIG,ROLLED,SLOPPILY
```

Spectrum owners must use speparate DIM statements in place of line 80 and shoul use single letter variables so that, for example, AD$ is A$. Also change 150 to:

```
150 LET R1=INT(RND*2):R2=INT(RND*10) etc.
```

VIC-20 Owners must use:

```
150 LET R1=INT(R(1)*2)+1 : LET R2=INT(RND(1)*10)+1 etc
```

Writing Your Own Programs

You are now in a position to start writing your own programs. Most beginners try to compose programs on the computer, typing in lines as they think of them.

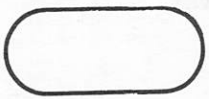
A better way is to spend some considerable time thinking about the algorithm or model which will make the computer do what you want. Draw up a flowchart to represent the algorithm. This preparation means that you will have to spend less time debugging, rewriting or renumbering programs. It also means that the algorithm can be used on different models of computer without the need to prepare it all over again.

Suppose you want to write a program which will make up addition sums and then tell the person using the program if the answer is right or wrong.

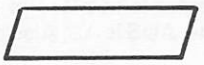
Begin by thinking of what has to be done. The computer needs:

to choose the numbers.
to print the sum.
to work out the answer to the sum.
to check the answer with that worked out by the computer.
to tell you whether you have got the answer right or wrong.

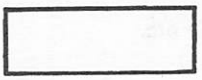
We use four different shaped boxes in flowcharting:



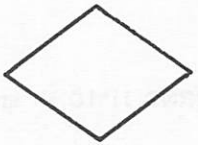
is used for Start and Stop



is used for Input and Output

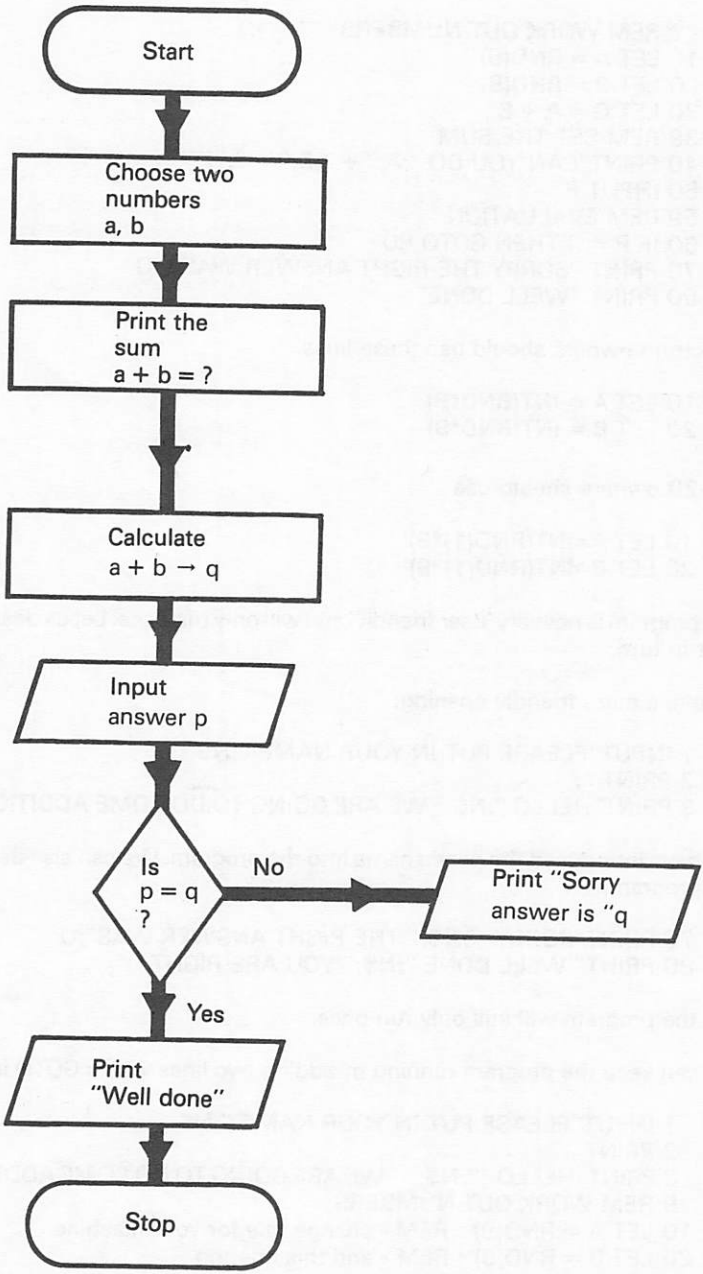


is used for Processing



is used for Questions

Here is the flowchart:



We now need to translate the flowchart into BASIC. We will use REM statements to help us to alter the program at a later date. The word REM is ignored by the computer - it will be printed when LISTed but otherwise has no effect on the program.

```
 9 REM WORK OUT NUMBERS
10 LET A = RND(9)
20 LET B = RND(9)
30 LET Q = A + B
39 REM SET THE SUM
40 PRINT"CAN YOU DO ";A;" + ";B;" = ";"?"
50 INPUT P
59 REM EVALUATION
60 IF P = Q THEN GOTO 80
70 PRINT "SORRY THE RIGHT ANSWER WAS ";Q
80 PRINT "WELL DONE"
```

Spectrum owners should use these lines:

```
10 LET A = INT(RND*9)
20 LET B = INT(RND*9)
```

VIC-20 owners should use

```
10 LET A=INT(RND(1)*9)
20 LET B=INT(RND(1)*9)
```

The program is not very 'user friendly' and will only run once. Let us deal with each fault in turn.

Here is a more friendly opening:

```
1 INPUT"PLEASE PUT IN YOUR NAME ";N$
2 PRINT
3 PRINT"HELLO ";N$;" WE ARE GOING TO DO SOME ADDITION SUMS.
```

We have introduced the user's name into the program. We can also use it later in the program:

```
70 PRINT"SORRY ";N$;" THE RIGHT ANSWER WAS";Q
80 PRINT" WELL DONE ";N$;" YOU ARE RIGHT.
```

But the program will still only run once.

We can keep the program running by adding two lines with a GOTO instruction.

```
 1 INPUT"PLEASE PUT IN YOUR NAME ";N$
 2 PRINT
 3 PRINT"HELLO  ";N$;"  WE ARE GOING TO DO SOME ADDITION SUMS.
 9 REM WORK OUT NUMBERS
10 LET A = RND(9) : REM - change this for your machine
20 LET B = RND(9) : REM - and this one too
30 LET Q = A + B
39 REM SET THE SUM
```

```
40 PRINT"CAN YOU DO ";A;" + ";B;" = ";"?"
50 INPUT P
59 REM EVALUATION
60 IF P = Q THEN GOTO 80
70 PRINT"SORRY ";N$;" THE RIGHT ANSWER WAS ";Q
71 GOTO 10
80 PRINT" WELL DONE ";N$;" YOU ARE RIGHT.
81 GOTO 10
```

Lines 71 and 81 instruct the computer to restart the program every time the answer to a sum has been given.

We will soon find that we can write quite a long program which works correctly. We can use fewer line numbers by joining lines together with a colon.

Programs Using STRINGS

A word processor is a program which will allow you to type text at the keyboard and then let you swap paragraphs around, change words and generally "process" what you have written until you are satisfied with the way it looks. The text can then be printed on a printer with no mistakes or crossings out.

Any word processing program will rely heavily on the manipulation of "strings". These are sequences of characters (letters and numbers) and are represented in a program by one or more letters followed by a $ sign. So A$ represents a string, so does B$. We can join strings together, like this:

```
10 LET A$ = " the cat "
20 LET B$ = " sat "
30 LET C$ = " on "
40 LET D$ = " the mat "
45 CLS : REM-VIC USERS PRINT "[CLR]
50 PRINT A$ + B$ + C$ + D$
51 PRINT
60 PRINT C$ + D$ + A$ + B$
61 PRINT
70 PRINT A$ + C$ + D$ + B$
71 PRINT
80 PRINT D$ + B$ + C$ + A$
```

In lines 10 to 40 each part of the sentence "The cat sat on the mat" is represented by the string variables A$, B$, C$ and D$. They are called 'variables' because we can change them easily - so, A$ can store *any* string.

Lines 50 to 80 print some amusing sentences.

Another thing that you can do with strings is to take the left hand, middle, or right hand parts of them. We do this with the BASIC string functions:

Spectrum owners do not have these commands, but look in your manual and you'll see how to do it, or look at the section on using DEF FN later in this chapter.

To illustrate the use of these functions, we will use three short programs.

The first two programs below need little explanation except to say that if you wanted the LEFTmost three letters of COMPUTER then PRINT LEFT$ ("COMPUTER",3) would print them. The RIGHTmost three letters would be printed if you typed PRINT RIGHT$("COMPUTER",3). In these programs the string of letters that you INPUT is used in the brackets so that they will work with any string (up to 255 characters).

```
10 PRINT
20 INPUT"TYPE A SENTENCE ";S$
30 PRINT
40 INPUT"HOW MANY LETTERS ";L
50 PRINT
60 PRINT"THE FIRST ";L;" LETTERS ARE :"
70 PRINT
80 PRINT LEFT$(S$,L)
90 PRINT
100 GOTO20
```

Spectrum owners insert:

```
75 DEF FN I$ (s$,L)=s$(To L)
80 PRINT FNL$(s$,L)
```

```
10 PRINT
20 INPUT"TYPE A SENTENCE ";S$
30 PRINT
40 INPUT"HOW MANY LETTERS ";L
50 PRINT
60 PRINT"THE LAST ";L;" LETTERS ARE :"
70 PRINT
80 PRINT RIGHT$(S$,L)
90 PRINT
100 GOTO20
```

Spectrum owners insert:

```
75 DEF FN r$(s$,n)=$s(L TO)
80 PRINT FNr$(s$,L)
```

MID$ is a little more complicated. If you wanted three letters from the MIDdle of the word COMPUTER then PRINT MID$("COMPUTER",4,3) would do it. The computer goes to the fourth letter and PRINTS the next three letters.

```
10 PRINT
20 INPUT"TYPE A SENTENCE ";S$
30 PRINT
40 INPUT"HOW MANY LETTERS ";L
50 PRINT
60 INPUT"FROM WHICH NUMBER LETTER ";N
70 PRINT
80 PRINT"THE ";L;" LETTERS FROM LETTER ";N;" ARE :"
90 PRINT
100 PRINT MID$(S$,N,L)
110 PRINT
120 GOTO20
```

Spectrum owners use

```
95 DEF FN m$(s$,n,L)=s$(nTOn+L-1)
100 PRINT FN m$(s$,n,L)
```

Programs Using LEN

LEN gives us the number of characters in a 'string'. It will count letters, numbers and spaces. Try this program:

```
10 LET A$ = "PETER PIPER"
20 PRINT LEN(A$)
```

The computer will print the number 11 because there are 10 letters and 1 space.

Here is a program that uses LEN to count the letters in a sentence. Line 60 works out how long the sentence is and then the FOR...NEXT loop in lines 70 to 90 checks along each letter of your sentence to see if you've typed any spaces or full stops. Then at line 110 the computer takes away the number of spaces and full stops and tells you how many letters there were in your sentence.

```
10 LET SP=0:LET FS=0
20 PRINT
30 PRINT"TYPE A SENTENCE"
40 PRINT
50 INPUT N$
60 LET L=LEN(N$)
70 FORI=1 TO L:IF MID$(N$,I,1)=" " THEN SP=SP+1
80 IF MID$(N$,I,1)="."THEN FS=FS+1
90 NEXT I
100 PRINT
110 PRINT"YOUR SENTENCE HAS ";L-SP-FS;" LETTERS"
120 PRINT
130 IF SP <> 0 THEN PRINT "YOU ALSO TYPED ";SP;" SPACES"
140 IF RIGHT$(N$,1)<>"."THEN PRINT"YOU FORGOT THE FULL STOP!"
150 GOTO 10
```

You could add a line to this program so that it could count how many Es (or any other letter of your choice) there were too. Can you sort out how to do it? Try:

```
5   LET E=0
85  IF MID$(N$,I,1)="E" THEN E=E+1
135 PRINT"YOU TYPED ";E;" E'S"
```

Programs Using DIM

The DIM or DIMension statement allows us to set up a series of "boxes" rather like those the Post Office uses when sorting letters. Each "box" will have a label attached to it so that the computer can keep track of where things go. Note, if we type DIM B(6), the computer will set aside 7 "boxes"! This is because the first is labelled B(0), the second B(1) and so on.

B(0) B(1) B(2) B(3) B(4) B(5) B(6)

We don't need to have just one number in a DIM statement. DIM N(10,10) for example, will instruct the computer to set up a table of 11 x 11 "boxes", making 121 in all. It is easy to run out of computer memory by DIMensioning very large arrays!!

The next program DIMensions an array as 11 x 11 and then, using FOR...NEXT loops, the computer will fill in each "box" with the product of the box's two "addresses". So N(6,3) will be filled with 6 x 3 = 18. This is exactly what you do when you fill in a table square. The program also allows you to check that each box is correctly filled.

```
10 DIM N(10,10)
20 FORI=1TO10
30 FORJ=1TO10
40 N(I,J)=I*J
50 NEXT J
60 NEXT I
70 INPUT"FIRST NUMBER ";X
80 INPUT"SECOND NUMBER ";Y
90 PRINT"BOX ";X;",";Y;" CONTAINS ";N(X,Y)
100 GOTO70
```

If this short program reminded you of the old game of battleships, well it was no accident! Here is a version of the game which uses a DIMension of 10 x 10 (Line 10). This time the "boxes" will contain words, so the DIMension statement will have variable N$, followed by the size.

Twenty battleships are positioned in the array (Lines 30 to 70) and you have to find them (Lines 90,100). If you type a number greater than 10 you will be told it is too high since that would be greater than the DIMensioned space (Line 110). You

are told the contents of a "box" and, if it contains battleship, a counter is increased (Lines 120,130). Once you have been to a "box" the computer will put "CHECKED" in it so that you will know if you have visited the same place before (Line 140).

```
10 DIM N$(10,10)
20 LET C=0:LET F=0
30 FOR I=1TO20
40 LET X=RND(10):LET Y=RND(10)
50 IF N$(X,Y)="A BATTLESHIP"THEN 40
60 LET N$(X,Y)="A BATTLESHIP"
70 NEXT I
80 PRINT
90 INPUT"FIRST NUMBER ";X
100 INPUT"SECOND NUMBER ";Y
110 IF X>10 OR Y>10 THEN PRINT"TOO HIGH":GOTO 80
120 PRINT"BOX ";X;",";Y" CONTAINS ";N$(X,Y)
130 IF N$(X,Y)="A BATTLESHIP"THEN F=F+1:PRINT"YOU'VE FOUND
    ";F;" BATTLESHIPS"
140 LET N$(X,Y)="CHECKED":C=C+1
150 PRINT
160 PRINT "YOU'VE HAD ";C;" TRIES"
170 PRINT
180 IF F=20 THEN GOTO 200
190 GOTO 90
200 PRINT"YOU'VE FOUND THEM ALL! AFTER ";C;" TRIES"
```

Spectrum users

```
40 LET X=INT(RND*10):LET Y=INT(RND*10)
```

VIC users

```
40 LET X=INT(RND(1)*10)+1:LETY=INT(RND(1)*10)+1
```

You could change the program to select fewer or more battleships. Change the 20s in lines 30 and 180. Or you could make the array smaller or larger. Change the 10's in lines 10,40,110.

Providing there is enough room in the computer's memory, we can DIMension many arrays. To return to the sorting idea, we could have an array for Names, one for Addresses, one for Telephone numbers etc. If we put all our information into these arrays, we could then get the computer to print out just what we need at that moment.

This is the idea of "information processing", something that the computer is very efficient at doing.

The program that follows is one that does something "useful"! It will allow you to keep track of your friends and relatives, their addresses, telephone numbers and birthdays. When you type it in to your computer, instead of the information

contained in the DATA lines, put your own. Make sure that you type the address with a *space* between the address number and road and *not* a comma.

Lines 10,40 DIMension arrays for the four pieces of information we are going to store.

Lines 50,70 READ the information contained in the DATA statements.

Line 80 INPUT what information you would like to find.

Line 85 ENDs the program if you INPUT the word END.

Lines 90,140 check all the arrays to see if they contain the information you want.

Line 150 is only used if the computer cannot find your information.

Lines 500,540 PRINT out all the information you need.

Lines 1000,1040 contain the DATA.

Once typed in you can RUN the program. If it works, it should be SAVEd onto a cassette tape. Then, each time you want to use the program you can LOAD it back in again.

When you RUN the program, type in either a NAME or an ADDRESS or a TELEPHONE number or a BIRTHDAY. The computer will try and match what you have typed with what is in its memory. If it is successful then *all* the details about that person will be PRINTed.

```
10 DIM NAME$(5)
20 DIM ADDRESS$(5)
30 DIM TEL$(5)
40 DIM BIRTHDAY$(5)
50 FOR I=1TO5
60 READ NAME$(I),ADDRESS$(I),TEL$(I),BIRTHDAY$(I)
70 NEXT I
75 PRINT
80 INPUT"SEARCH FOR ";SEARCH$
85 IF SEARCH$="END"THEN END
90 FOR I=1TO5
100 IF SEARCH$=NAME$(I) THEN 500
110 IF SEARCH$=ADDRESS$(I) THEN 500
120 IF SEARCH$=TEL$(I) THEN 500
130 IF SEARCH$=BIRTHDAY$(I) THEN 500
140 NEXT I
145 PRINT
150 PRINT"NOT FOUND":GOTO 80
500 PRINT
505 PRINTNAME$(I)
```

```
510 PRINT"ADDRESS ";ADDRESS$(I)
520 PRINT"TELEPHONE ";TEL$(I)
530 PRINT"BIRTHDAY ";BIRTHDAY$(I)
540 PRINT
550 GOTO 80
1000 DATA ROBERT,5 HIGH STREET,25465,JULY 11
1010 DATA TIMOTHY,36 DALE ROAD,87987,JUNE 28
1020 DATA RACHEL,43 LEWIS STREET,32172,MAY 11
1030 DATA ANNA,76 BROW HILL,90324,NOVEMBER 10
1040 DATA PAT,30 FAIRFIELD,10298,JUNE 17
```

SPECTRUM + VIC owners should only use single letter variables:

e.g. NAME$ (I)
  becomes N$(I)
ADRESS$(I)
  becomes A$(I)
etc

As you increase the number of people listed in the DATA statements, so you will need to change lines in the program. As the program stands there are details of five people stored in it. If you increased this to 25 then all the 5s in the program (Lines 10,20,30,40,50,90) would have to be changed to 25.

Another amendment you might like to make is to include another DIMension for a further section of information, like COMMENTS. You would then have to alter several lines like this:

```
45 DIM COMMENTS$(5)
65 READ COMMENTS$(I)
535 PRINT"COMMENTS ";COMMENTS$(I)
1000 DATA ROBERT,5 HIGH STREET,25465,JULY 11,LIKES STAR WAR
MODELS
```

Programs Using GET$ (INKEY$ for Spectrum owners)

GET$ (or INKEY$ on a Spectrum) is unlike INPUT, because it does not wait for RETURN or ENTER. It is very useful in games because not only is it quicker than the INPUT method, it does not print anything on the screen.

When the computer reaches the GET$ (Spectrum INKEY$) it searches the keyboard to see if any key has been pressed and then stores it in memory. If no key is pressed a null string (i.e. an empty string) is taken into the program.

Try this program to see it in use.

```
10 PRINT AT5,5;"PRESS ANY KEY"
20 PAUSE 200: CLS
```

```
30 PRINT AT 11,10,INKEY$
40 PAUSE 100
50 GO TO 10
```

VIC changes:

```
10 GETA$:IFA$= ""THEN 10
20 PRINT"You PRESSED ";A$
30 GOTO 10
```

BBC changes

```
10  A$=GET$
20  PRINT "You PRESSED ";A$
30  GO TO 10
```

Programs Using DEF FN

The Spectrum can work out some functions directly. We can also put in user defined functions using DEF FN which means DEFine FuNction.

Suppose we put in:

```
 5 LET N=3
10 DEF FN F(N) = N*N
20 PRINT N
30 PRINT FN F(N)
```

The computer works out the function of N in line 10. Line 20 prints out the original value of N and line 30 the function of N.

Mark Harrison, author of "The Sinclair Spectrum in focus" (also published by Sigma) shows the use of DEF FN to overcome the Spectrum's lack of the BASIC statements LEFT$, MID$ and RIGHT$.

Define each as a FUNction:

LEFT$ becomes DEF FN l$(a$,n) = a$ (TO n)
MID$ becomes DEF FN m$(a$,m,n) = a$ (m TO m + n-1)
RIGHT$ becomes DEF FN r$(a$,n) = a$ (n TO)

Try this short program to see the DEF FN solution at work:

```
10 LET a$="CONSTANTINOPLE"
20 DEF FN l$(a$,n)=a$( TO n)
30 PRINT FN l$(a$,3)
40 DEF FN m$(a$,m,n) =a$(m TO m+n-1)
50 PRINT FN m$(a$,4,5)
60 DEF FN r$(a$,n) =a$(n TO )
70 PRINT FN r$(a$,6)
```

Here is an example of DEF FN on the BBC

```
10 DEF FNMULT(N)=(42*N)/6
20 INPUT "TYPE A NUMBER "N
30 PRINT "THE FORMULA GIVES ";FNMULT(N)
40 GOTO20
```

And here is the VIC-20 version:

```
10 DEF FNA(N)=(42*N)/6
20 INPUT "TYPE A NUMBER "N
30 PRINT "THE FORMULA GIVES ";FNA(N)
40 GOTO20
```

When your programs (Gasp) DON'T WORK? Read on .....

Sometimes, our carefully composed programs fail to do what we want and, often, programs laboriously copied from magazines or books fail to work. This section might help you to cope when things go wrong.

There are two main sorts of error:

Computer indicated errors which prevent the program from running

Design errors, i.e. when the program runs but gives the wrong results.

We'll look at each of these in detail:

Computer indicated errors

The computer is able to identify errors that do not obey the rules of BASIC. In some cases (e.g. on the Spectrum) it does this as the line is typed in, but in others it will cause your program to 'crash'. In either case, the computer will indicate the error either by a short description or by an error code.

The simplest procedure is to find the description of the error in the manual for your computer, and then to correct it.

If this doesn't work it is likely that the original program design is wrong. So it's back to the drawing board and flow charts.

Let us assume, however, that the design seems correct and so we are unable to make a change. Try running the program without the offending line. It is a good idea to put this line into a REM statement. The computer, hopefully, will give an indication of what the deleted line was intended to do-this will usually be a computer indicated error.

The computer may run the program without the line, although, of course, something will be missing from the display or result. By comparing the indicated error caused by the offending line to the error indicated by its omission, it is often possible to find out what the line was intended to do.

The computer manual, or one of the books designed to go one step beyond the manual, will suggest a way of writing a line or lines which will perform the intended task. Compare this new line, or lines, to the original one which caused all the trouble. They may be similar or substantially different.

Put the new line into the program and try running it. Hopefully this will have solved the problem. If it does not, there is probably an error in another line. The most common of these errors arise in GO TO, GOSUB or FOR/NEXT loops. Search the program and check that each is as it should be.

The GO TO instruction must jump to the correct line. The GOSUB routine must not alter variables so that they cannot be used later in the program.

The FOR/NEXT loop must use the same variables in the FOR and NEXT lines.

IF all this fails to make the program run successfully, the best course is probably to devise a new algorithm and start the process again.

Design Errors

The first step is to check that the program has been fully and correctly entered and, secondly, that it is capable of doing what you require.

Assuming no such faults are found, it is now time to examine the way in which variables are being changed or used in the program. Some versions of BASIC (e.g. BBC BASIC) have a command TRACE which will show you how the variable is being used in the program. Unfortunately, the Spectrum and VIC do not have the TRACE command although 'chips' are available with it on.

So, supposing the TRACE command is unavailable try putting PRINT statements at every step in the program so that the value of the variable can be checked. Each checking line can be given an identifier so that it is easily found. This process is shown in the following simple example.

Untraced	**Traced**
10 LET A = 4	10 LET A = 4
20 LET B = 6	11 PRINT"****";A
30 LET X = A * B	20 LET B = 6
40 PRINT X	30 LET X = A * B
50 END	40 PRINT X
	41 PRINT"****";X,A,B
	50 END

It is now a simple process to check the variables as the program runs. An alternative and shorter method is to place GO TO or GOSUB statements around those parts of the program which are known to be O.K. This will let you concentrate on the problem area of the program.

Do this by using PRINT statements to see if the variables are being processed as expected. If they are being correctly processed but an irrelevant output still results, the flowchart must be incorrectly drawn or else it is incorrectly translated into BASIC.

If the flowchart is correctly translated into BASIC but fails to work it must be drawn wrongly. The best solution to this sort of problem is to find a method which will carry out the task. There are many programs and books available and one of them is sure to have a subroutine which will do what is wanted.

Coping with difficulties like this is one of the best ways of learning to program. Our advice is to stick at the problem until you have beaten it.

STRUCTURED PROGRAMMING - a way to make your programs work first time!

Throughout this chapter and generally through the book we encourage you to adopt the habit of structured programming. One of the criticisms made of BASIC is that it is too loosely structured. In many ways BASIC is the wrong language to start off with, because of this lack of structure. We have to unlearn many of the bad habits we have got into using BASIC if we want to learn a structured language like Pascal. Pascal is a highly structured language which demands strict attention to its rules. A missing ';' stops the program running altogether. So we need to prepare you for these stricter requirements, and will do this by looking at structured programming.

Let us suppose that we have a problem to solve using the microcomputer and decide to use the structured programming approach. There are three stages on the road to the structured program:

We will look at these in turn.

The **Analysis** stage is where we decide what we actually want the program to do. It is helpful to think of each program as having three elements:

the inputting of data
the efficient processing of that data
the production of useful output

Careful thought at this stage can save lots of wasted time and effort later.

The **Design** stage puts the thoughts of the Analysis stage into a structure which

works from beginning to end. There are two popular ways of doing this:

The **top down** design looks at the whole ultimate aim/purpose of the program and then divides this into major subroutines (or procedures for BBC users). These large subroutines are then subdivided into smaller routines and these in turn are again divided into parts of subroutines. We will eventually have the program structured in a number of small parts which are all related to the whole program idea.

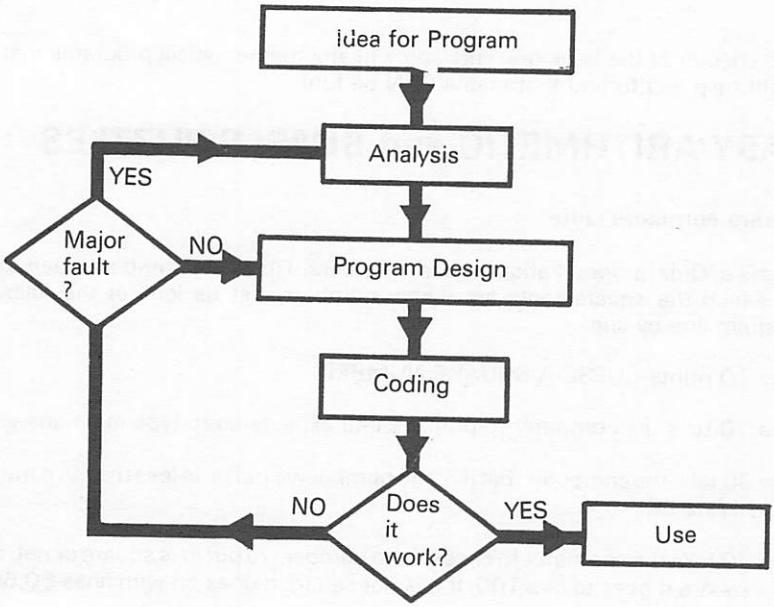
The **bottom up** design works the opposite way round. We start with the smallest tasks of the program - parts of subroutines. They are linked to other small tasks to make a bigger part of the program. These bigger parts are then linked together to form large sections of the program. We keep on doing this until all parts of the program have been linked together.

The program has now been reduced to a series of subroutines (procedures for BBC users). All the parts of the program have been thought about and arranged in a logical order. It is now an easy task to find the subroutine which is wrong and to replace it. Probably the worst fault is to use lots of GO TO instructions. This makes a program slow to run and very difficult to change a later date.

We now have to Code the results in BASIC language. Here a library of subroutines is very useful. There are lots of subroutines in this book. A good idea, when you are starting to write your own programs, is to look at a program which does the same sort of job as you want your program to do. You can often take whole subroutines out of one program to use in other programs.

Try to make sure that your program runs from start to finish and does not need to jump about from place to place. As we said before, the fewer the number of GO TOs used the better. Look at the flowcharts in this book and try to follow the ideas of the program writer - why was one way chosen rather than another ? Perhaps you can improve on the examples shown.

This flowchart shows how structured programming takes place.


Idea for Program
Analysis
YES
Major fault ?
NO
Program Design
Coding
NO
Does it work?
YES
Use

CHAPTER 3

Some Mathematical Programs

This chapter of the book presents some useful mathematical programs that just might help you to find that maths CAN be fun!

EASY ARITHMETIC and SIMPLE PUZZLES

Square Numbers Quiz

This is a 'Quiz' program about square numbers. These are numbers, such as 25, for which the square roots are whole numbers. Let us look at the following program line by line.

Line 10 prints GUESS A SQUARE NUMBER.

Line 20 tells the computer to print a ? and expects us to type in an answer.

Line 30 tells the computer that if S, the number we put in, is less than 0, it must go to line 80.

Line 40 tells the computer to check if the number we put in is square or not. If it is square it goes to line 100, if it is not square it goes on with lines 50,60 and 70.

Line 50 tells us that our answer is not a square number.

Line 60 asks us to make another guess.

Line 70 tells the computer to go to line 20 so that another guess can be made.

Line 80 tells us that we chose a number less than 0.

Line 90 tells the computer to go to line 60 and so another guess is asked for.

Line 100 tells us that our guess was correct.

Line 110 tells the computer to go to line 10 and so start the program again.

```
NEW
10 PRINT"GUESS A SQUARE NUMBER.
20 INPUT S
30 IF S<0 THEN 80
40 IF SQR (S)=INT (SQR(S)) THEN 100
50 PRINT "TRY AGAIN".;S;" IS NOT SQUARE.
60 PRINT "WHAT IS YOUR NEXT GUESS?"
70 GO TO 20
80 PRINT "SORRY, ";S;" IS LESS THAN 0"
90 GO TO 60
100 PRINT "WELL DONE"
110 GO TO 10
120 END
RUN
```

Guess the Number

In this program, the computer will choose a number, in line 50, that is between 1 and 100. It will keep this number a secret and you have to try and guess what it is.

The instructions are given in lines 20,30 and 40. The blank PRINT lines 11,21,31 and 41 will leave spaces between lines 10,20,30 and 40 making them easier to read.

Line 70 accepts your guess and it is compared to the computer's secret number in lines 80,90 and 100. If the guess is correct, the program goes to line 110. If it is too big, then you jump to 150 or, if too small, to line 170. You do this by using IF and THEN on these lines.

The GO TO instruction in line 140 ends the program unless RUN is typed in line 130. The GO TO instruction in line 210 makes sure more guesses are allowed by returning you to line 70.

Lines 60,110 and 200 keep a count of how many guesses you have had. Line 60 sets the counter at 0. Lines 110 and 120 add one for each guess we have.

If your guess is right in line 80, the computer goes to line 110 where one is added to the counter. Congratulations are printed by line 120.

```
NEW
10 INPUT"TYPE IN YOUR NAME";N$
11 PRINT
20 PRINT"WELL ";N$;" I WILL CHOOSE A NUMBER"
21 PRINT
30 PRINT"BETWEEN 1 AND 100"
31 PRINT
40 PRINT"YOU TRY TO GUESS IT.
41 PRINT
50 LET N=INT(RND(0)*99)
```

```
60 LET K=0
70 INPUT G
80 IF G=N
90 IF G>N THEN 150
100 IF G<N THEN 170
110 LET K=K+1
120 PRINT"WELL DONE ";N$" GOT IT RIGHT IN ";K;" TRIES.
130 PRINT"TO PLAY AGAIN TYPE 'RUN'.
140 GO TO 220
150 PRINT"SORRY ";N$;" GUESS WAS TOO HIGH.
160 GO TO 180
170 PRINT"SORRY ";N$;" GUESS WAS TOO LOW.
180 PRINT
190 PRINT"PLEASE TRY AGAIN. GOOD LUCK!"
200 LET K=K+1
210 GO TO 70
220 END
RUN
```

VIC users should change line 50 to:

```
50 LET N=INT(RND(1)*100)+1
```

And, BBC users will need:

```
50 LET N=RND(100)
```

Here is a program to print out a multiplication table:

```
10 PRINT "THIS PROGRAM WILL WORK OUT ANY"
20 PRINT "MULTIPLICATION TABLE FOR US.
30 INPUT "WHICH TABLE DO YOU WANT?";N
40 LET C=0
50 LET P=N*C
60 PRINT N;"X";C;"=";P
70 LET C=C+1
80 IF C > 20 THEN 100
90 GO TO 50
100 PRINT "END"
```

Lines 10,20 give instructions. We are asked to put in the number of the table we want in line 30. Line 40 tells the computer to start at 0 times the number we put in as N in line 30. Line 50 works out the answer to each of the multiplication sums which make up the table.

Line 60 tells the computer to print the sum it worked out in line 50. Line 80 tells the computer to go to line 100 when it reaches 20 times the number we chose as N. Line 90 tells the computer to go back to the line where the multiplication sum is worked out.

Line 100 prints END after the program has worked out 20 x N.

Factors

Now to a program which will work out factors of any number we choose.

```
10 PRINT"PUT IN THE NUMBER TO BE FACTORED"
11 INPUT N
12 CLS:REM VIC NEEDS PRINT "[CLR]"
20 FOR I = 1 TO N
30 IF N/I=INT(N/I) THEN PRINT I
40 NEXT I
49 PRINT"  "
50 PRINT"ARE THE FACTORS OF ";N
51 PRINT
60 GO TO 10
```

Line 10 asks us to put in the number for which we require the factors. The computer will remember this number as N. Line 12 clears the screen so that the finished display looks better.

Line 20 asks the computer to calculate a number, I, from 1 to the number it remembers as N.

Line 30 divides N by I and checks if this is the same answer as N divided by I when the answer is a whole number. If it is, then that number is printed as I. INT stands for INTEGER which is a whole counting number (not a fraction or a decimal).

Line 40, being the NEXT part of a FOR/NEXT routine, makes sure that N in line 30 is divided by 1 then by 2 then 3 and so on up to whatever number N is. Lines 49,50 and 51 produce a nice layout for the answers.

Line 60 let us re-use the program if we wish to.

Prime Numbers

Factors are closely related to prime numbers and here is a program which will print the prime numbers up to any limit we wish to set.

```
10 PRINT"PUT IN THE UPPER LIMIT"
11 INPUT L
12 CLS:REM VIC NEEDS PRINT "[CLR]"
20 FOR I = 2 TO L
30 LET J = 2
40 LET A = J*INT(I/J)
50 IF A = I THEN GO TO 100
60 LET J = J+1
70 IF J=I THEN GO TO 90
```

```
80 GO TO 40
90 PRINT I;
100 NEXT I
```

Lines 10 to 12 ask us to put in the highest number we want to look at.

Line 12 clears this off the screen. As 2 is a prime number itself we need to isolate it from the range of numbers to be examined. This is done in line 20. The only even prime number is 2 so we can discard all other even numbers. This is done in line 40. Line 40 is the most important in the program. Try taking some numbers through it to see how it works. Line 60 keeps on adding 1 to the previous number known as J which started as 2 in line 30.

When this program is RUN the prime numbers appear rather slowly because of the complicated instruction in line 40 and because of the number of values of "I" that must be examined. Where there is a big gap in the series of primes, the lapse in time is noticeable.

Mathematical Patterns

Now to some closely related programs which will work out number patterns for us. Enter the program and RUN it.

```
10 INPUT N
11 LET C=N
12 LET I=0
30 LET A = N*N
40 PRINT N;" X ";N;" = ";A
50 LET N = (N*10)+C
51 LET I = I+1
52 IF I>5 GO TO 70
60 GO TO 30
69 PRINT
70 PRINT " HOW'S THAT FOR A NICE PATTERN?"
```

Lines 12 and 52 stop the program running on and on. By adding line 20 and changing lines 30 and 40 to include the variable T, we get:

```
10 INPUT N
11 LET C=N
12 LET I=0
20 LET T=N - 1
30 LET A=N*T
40 PRINT N;" X ";T;" = ";A
50 LET N=(N*10)+C
51 LET I=I+1
52 IF I>5 GOTO 70
60 GOTO 20
```

```
69 PRINT
70 PRINT "HOW'S THAT FOR A NICE PATTERN?"
```

By deleting line 20 and changing lines 30,40 and 52 we produce a different pattern of numbers.

```
10 INPUT N
11 LET C=N
12 LET I=0
30 LET A=N*C
40 PRINT N;" X ";C;" = ";A
50 LET N=(N*10)+C
51 LET I=I+1
52 IF I>8 THEN GOTO 70
60 GOTO 30
69 PRINT
70 PRINT"HOW'S THAT FOR A NICE PATTERN?"
```

Using similar alterations and modifications you can produce a variety of number patterns.

Squares and Circles

Here is a short program that will work out various properties of a square, and of a cube for which the square forms one face. Can you see how it works?

```
10 INPUT "Length of the square side ";l
11 LET p=4*l
19 PRINT
20 PRINT "Perimeter of the square is ";p
21 LET a = l*l
29 PRINT
30 PRINT "Area of the square is ";s;" square units.
31 LET s = 6*a
39 PRINT
40 PRINT "Surface area of the cube is ";s;" square units.
41 LET v = l*l*l
49 PRINT
50 PRINT "Volume of the cube is ";v;" cubic units.
```

Here is another similar program which works out various calculations connected with a circle.

```
10 INPUT "Radius of the circle ";r
11 LET p =2*PI*r
19 PRINT
20 PRINT "Perimeter of the circle is ";p
21 LET a = PI*r*r
```

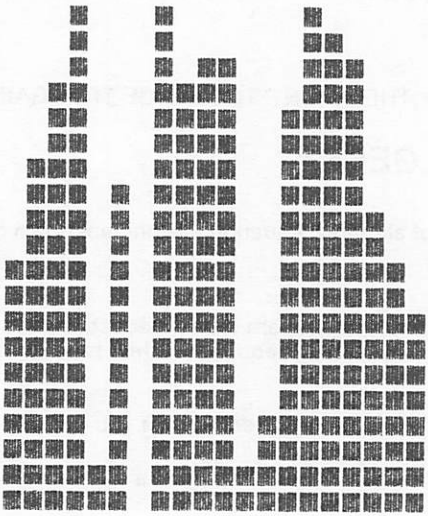
```
29 PRINT
30 PRINT "Area of the circle is ";a;" square units.
31 LET  s = 4*a
39 PRINT
40 PRINT "Surface area of sphere is ";s;" square units.
41 LET v =(4/3)*a
49 PRINT
50 PRINT "Volume of the sphere is ";v;" cubic units.
```

HISTOGRAMS

This program will draw a histogram from the data that you input. The program begins by asking you for the type of computer you are working on - this is to ensure the display is drawn correctly. Then you will be asked for the maximum values for the X and Y axes, followed by the data for each X value.

When this is completed, the histogram will be drawn by the computer, similar to the figure shown below:

```
10 REM PUT CLEAR SCREEN HERE
20 INPUT"BBC, VIC OR SPECTRUM ";Q$
30 IF Q$="BBC"THEN LET C=255: LET  W=40:GOTO 60
40 IF Q$="VIC"THEN LET C=166:LET W=22:GOTO 60
50 LET C=143:LET W=32
60 INPUT"HIGHEST Y VALUE ";Y
70 INPUT"HIGHEST X VALUE ";X
80 FORS=1TO(W-X)/2:S$=S$+" ":NEXTS
90 DIM G$(X,Y)
100 FORI=TOX:FORR=1TOY:G$(I,R)=" ":NEXTR,I
110 FORI=1TOX
120 PRINT"Y VALUE FOR X=";I;:INPUT V
130 IF V=O THENGOTO150
140 FORR=1TOV:G$(I,R)=CHR$(C):NEXTR
150 NEXTI
160 REM PUT CLEAR SCREEN HERE
170 FORI=Y TO 1 STEP-1
180 FORR=1TOX:H$=H$+G$(R,I):NEXT R
190 PRINTS$;H$
200 LET H$=""
210 NEXT I
```



The program works by storing your data in a matrix (this is just like a 'table of values'), which is defined by the DIM statement. When all data has been entered, the computer simply scans the matrix, from left to right, printing out what it finds in each of the cells.

Line 80 works out how many spaces to print in order to centralise the display.

Line 90 sets up the matrix.

Line 100 fills the matrix with spaces.

Lines 110-150 gets your data and fills the correct cells of the matrix with the plot character.

Lines 170-210 prints the matrix from the top down.

As you can see from the screen print for the BBC computer the axes are neither labelled nor scaled. Perhaps you could introduce these features as well as a title line into the program.

Other important omissions in the program are "mug-traps" to throw out illegal input values. Run the program and experiment a little by putting in values which will either cause the computer to drop out of the program with an error message or mess up the screen display. Now try to remedy these "bugs" by inserting lines after the INPUTs which will prompt the user to try again. For example, line 65 could be

```
65 IF Y>20THEN PRINT"TOO LARGE TRY AGAIN":GOTO 60
```

Line 75 could be:

```
75 IF X>(W-1)THEN PRINT"TOO LARGE TRY AGAIN":GOTO70
```

SIMPLE ALGEBRA

The simplest sort of algebraic equation has one unknown quantity. For example,

$a + 3 = 5$

It is hardly worth writing a program to solve equations like this but in doing so it can help us to solve more useful equations. Here is a program to solve equations like $a + 3 = 5$.

```
10 PRINT "Put in b from the equation a + b = c"
15 INPUT b
20 PRINT " Put in c from the equation a + b = c"
25 INPUT c
30 LET a = c - b
40 PRINT " The unknown number is"; a
```

If the equation has a minus sign like this: $a - b = c$, we will need to change line 30 to 30 LET $a = c + b$, or, even easier, just type the number 'b' with a minus sign in line 15.

The same sort of equation might have two unknowns, like this:

$a + b = 7$

There is an infinite number of solutions to this equation so we need a program which will accept correct solutions and throw out incorrect ones. In other words, it will see if *YOUR* answer is O.K., like this:

```
10 PRINT " Put in a from the equation a + b = c"
11 INPUT a
20 PRINT " Put in b from the equation a + b = c"
21 INPUT b
30 PRINT " Put in the answer to the equation"
31 INPUT c
40 IF a = c - b THEN PRINT" Solution is correct "
50 IF a <> c - b THEN PRINT" Sorry solution is incorrect"
```

Remember to type in the numbers with their correct signs.

Solving an equation like:

$2a + 3 = 7$

is really easy:

```
10 PRINT"Put in b from the equation 2a + b = c "
15 INPUT b
20 PRINT"Put in c from the equation 2a + b = c "
25 INPUT c
30 LET a = c - b
40 LET a = a / 2
50 PRINT a
```

Now try to write a program which will test whether the numbers you input will solve the equation $a^2 + b = c$. When your program works, change it to test answers for the equation $a^2 + b^2 = c$.

Remember that equations like these have many possible correct answers.

SIMULTANEOUS EQUATIONS

In general, we can write such equations as:

$$ax + by = c$$
$$px + qy = r$$

In order to find the values of x and y that simultaneously solve these two equations, we do some crafty mathematics. We start by multiplying the first equation by "q" and the second equation by "b" to give:

$$aqx + bqy = cq$$
$$bpx + bqy = br$$

Then we subtract one from the other, eliminating y, and find x:

$$aqx - bpx = cq - br$$

or

$$x = (cq - br)/(aq - bp)$$

And you do a similar thing to find y.

Of course, that's a bit like hard work, so here's a program to do it for you:

```
10 INPUT "Put in 'a' from equation 1 ";a
11 INPUT "Put in 'b' from equation 1 ";b
12 INPUT "Put in the answer 'c' to equation 1 ";c
13 INPUT "Put in 'p' from equation 2 ";p
14 INPUT "Put in 'q' from equation 2 ";q
15 INPUT "Put in the answer 'r' to equation 2 ";r
20 LET t=a*q-b*p
```

```
30 LET x=( c*q - b*r )/t
40 LET y=( r*a - c*p )/t
45 PRINT "The solution to these two simultaneous equations is :"
46 PRINT
50 PRINT "x is ";x
60 PRINT "y is ";y
```

FERMAT's Last Theorem

A French mathematician, Pierre de Fermat, reckoned that, although equations like

$$x^2 + y^2 = z^2$$

have several solutions such as

$$3^2 + 4^2 = 5^2$$

equations like

$$x^3 + y^3 = z^3$$

could not be solved, except by using zeroes.

Mathematicians have been trying to prove him wrong for many years. This program might help you to see if Fermat was right.

```
10 PRINT "Pierre de Fermat thought that"
11 PRINT
12 PRINT "there could be no solution to"
13 PRINT
14 PRINT " equations like:"
15 PRINT
16 PRINT " X*X*X + Y*Y*Y = Z*Z*Z
17 PRINT
18 PRINT "Let us test his theory."
19 PRINT
20 PRINT "Put in the values for X,Yand Z"
21 PRINT
22 INPUT "Put in X";X
23 INPUT "Put in Y";Y
24 INPUT "Put in Z";Z
30 LET a=X*X*X+Y*Y*Y
31 LET b=Z*Z*Z
32 CLS
33 BORDER 1 : REM Lines 33 and 34 only on Spectrum
34 PAPER 6
40 IF a<>b THEN PRINT AT 10,4; "Fermat was right this time."
41 If a=b THEN PRINT AT 10,4;"Fermat was wrong !"
```

After trying it for some numbers that you think might work try increasing the equation to:

$x^4 + y^4 = z^4$

If you do find a set of numbers which works-let us know!

Quadratic equations

You've probably had to solve many equations like this:

$ax^2 + bx + c = 0$

Here's a program to make it easy! See if you can spot the really important lines.

```
10 CLS
20 PRINT"QUADRATIC EQUATION SOLVER"
30 PRINT"-------------------------"
40 PRINT
50 PRINT"FOR EQUATIONS OF THE TYPE"
60 PRINT
70 PRINT "  A*X*X + B*X + C = 0"
80 PRINT
90 PRINT"THE SOLUTION IS FOUND BY USING"
100 PRINT"THE QUADRATIC FORMULA :"
110 PRINT
120 PRINT"       + ------------"
130 PRINT" X = -B - / B*B - (4AC)"
140 PRINT"
150 PRINT"           2A"
160 PRINT:PRINT
170 INPUT"X SQUARED COEFFICIENT (A) ";A
180 INPUT"        "COEFFICIENT (B) ";B
190 INPUT"          CONSTANT (C) ";C
200 LET D=B*B-4*A*C
210 IF D<0 THEN GOTO 290
220 PRINT
230 LET D=INT(1000*SQR D)/1000
240 LET XA=INT(1000*(-B+D)/(2*A))/1000
250 LET XB=INT(1000*(-B-D)/(2*A))/1000
260 PRINT
270 PRINT"  X = ";XA;" OR ";XB
280 GOSUB 360:RUN
290 PRINT
300 PRINT"THE EQUATION ";A;"X*X + ";B;"X = ";C
310 PRINT"HAS NO REAL ROOTS"
320 PRINT"BECAUSE B*B - 4AC = ";D
330 PRINT"AND THE SQUARE ROOT OF ";D
```

```
340 PRINT"IS IMAGINARY"
350 GOSUB 360:RUN
360 PRINT:PRINT"ANOTHER (Y/N) ";
370 INPUT A$:IF A$="Y"THEN RETURN
380 END
```

Simple trigonometry

Most BASICs include the trigonometric functions SIN, COS and TAN. Others may include ATN, ASN and even RAD and DEG for conversion between radians and degrees. Most of these functions are used with right-angled triangles. This short program tests the lengths of a triangle and then tells you if it is right angled or not.

```
10 INPUT"FIRST SIDE ";S1
20 INPUT"SECOND SIDE ";S2
30 INPUT"HYPOTENUSE ";H
40 IF H*H<>(S1*S1)+(S2*S2) THEN PRINT"NOT ";
50 PRINT"RIGHT ANGLED!"
```

Notice that the program uses a "default" method. Line 40 tests for Pythagoras' condition and will print "NOT" if it is not satisfied. Line 50 prints "RIGHT ANGLED!" whatever kind of triangle is found.

Before running the next program, owners of the BBC machine must type in @%=&20406 and RETURN so that the machine will print the correct number of decimal places.

The computer needs all angles to be specified in radians, not degrees. Since 1 radian=180/pi we must include a conversion factor of 57.296. This program prints out the sine, cosine and tangent of any inputted angle.

```
10 INPUT"ANGLE IN DEGREES ";A
20 LET C=57.296
30 PRINT"SINE    =";SIN(A/C)
40 PRINT"COSINE  =";COS(A/C)
50 PRINT"TANGENT =";TAN(A/C)
```

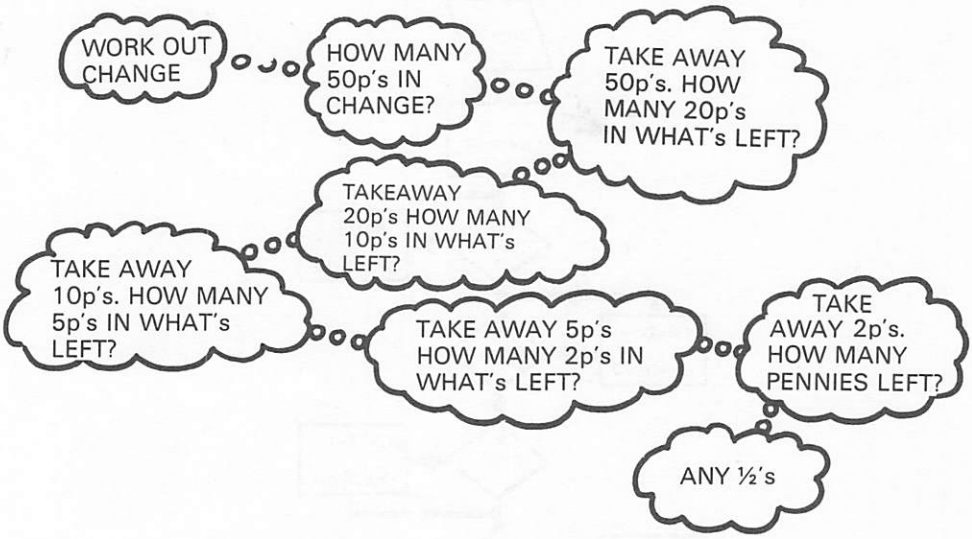
An extension to these programs could be to calculate all the sides and angles from input values for three sides, or two sides and one angle. Owners of BBC or Spectrum machines could even program the computer to draw the triangle.

"CHANGEGIVER" - a program design example

Have you been in a supermarket where the cashier has an automatic change dispenser ? This device allows the cashier to type in the total sales and the money tendered, but the change given is worked out automatically (except for £ notes!!).

Let's analyse the situation first and use "clouds" to design our program. Then we can think of variables, draw a flow chart and finally write the program.

Here is an outline in clouds:

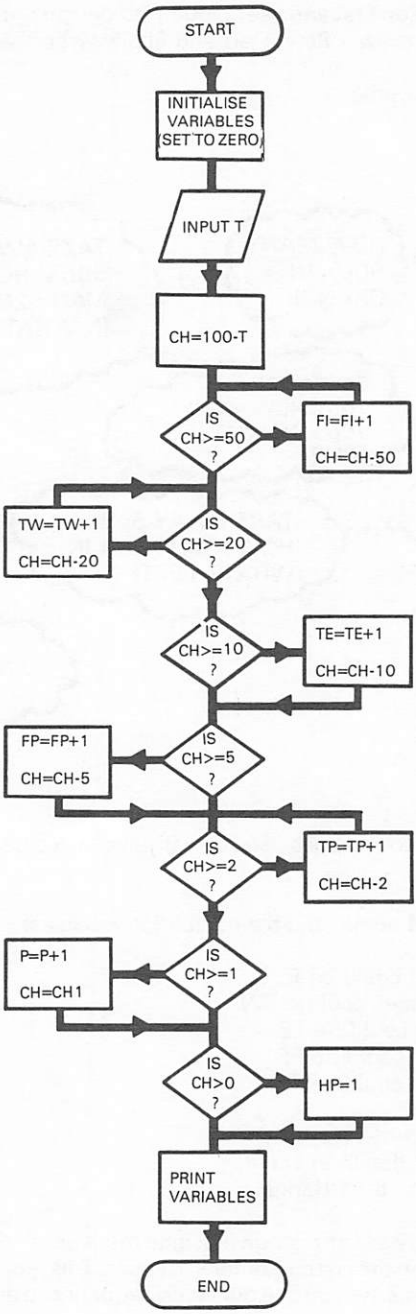

WORK OUT CHANGE
HOW MANY 50p's IN CHANGE?
TAKE AWAY 50p's. HOW MANY 20p's IN WHAT's LEFT?
TAKEAWAY 20p's HOW MANY 10p's IN WHAT's LEFT?
TAKE AWAY 10p's. HOW MANY 5p's IN WHAT's LEFT?
TAKE AWAY 5p's HOW MANY 2p's IN WHAT's LEFT?
TAKE AWAY 2p's. HOW MANY PENNIES LEFT?
ANY ½'s

The 1/2p is just about to disappear, so you will have to decide whether to include this or not.

And now, let's think of some suitable names for variables:

FIfty pence pieces could be FI.
TWenty pence pieces coul be TW.
TEn pence pieces could be TE.
Five Pence pieces could be FP.
Two Pence pieces could be TP.
Pennies could be P.
Half Pennies could by HP.
T can be used for the Total spent.
CH can be used for the CHange.

Now, before each purchase the change machine must be set to zero to make sure there is nothing left from the last customer's account. This is called "initialising the variables". Following this, we can use our cloud sequence to draw up a flow chart:


START
INITIALISE VARIABLES (SET TO ZERO)
INPUT T
CH=100-T
IS CH>=50 ?
FI=FI+1
CH=CH-50
IS CH>=20 ?
TW=TW+1
CH=CH-20
IS CH>=10 ?
TE=TE+1
CH=CH-10
IS CH>=5 ?
FP=FP+1
CH=CH-5
IS CH>=2 ?
TP=TP+1
CH=CH-2
IS CH>=1 ?
P=P+1
CH=CH1
IS CH>0 ?
HP=1
PRINT VARIABLES
END

The program now almost writes itself!!

```
10 LET FI=0:LET TW=0:LET TE=0:LET FP=0:LET TP=0:LET P=0:LET
   HP=0
20 INPUT"AMOUNT SPENT ";T
30 LET CH=100-T
40 IF CH>=50 THEN FI=FI+1:CH=CH-50:GOTO40
50 IF CH>=20 THEN TW=TW+1:CH=CH-20:GOTO50
60 IF CH>=10 THEN TE=TE+1:CH=CH-10
70 IF CH>=5 THEN FP=FP+1:CH=CH-5
80 IF CH>=2 THEN TP=TP+1:CH=CH-2;GOTO80
90 IF CH>=1 THEN P=P+1:CH=CH-1
100 IF CH>0 THEN HP =1
110 PRINT
120 PRINT FI;" FIFTY PENCE"
130 PRINT TW;"TWENTY PENCE"
140 PRINT TE;"TEN PENCE"
150 PRINT FP;"FIVE PENCE"
160 PRINT TP:"TWO PENCE"
170 PRINT P;"PENNY"
180 PRINT HP;" HALFPENNY"
190 PRINT:PRINT"AMOUNT SPENT = ";T
200 PRINT"CHANGE GIVEN = ";100-T
220 END
```

Try amending the program to give change from £5.

You could also put in a few lines so that the program will calculate the change from any amount (including the number of £1 notes or £1 coins!).

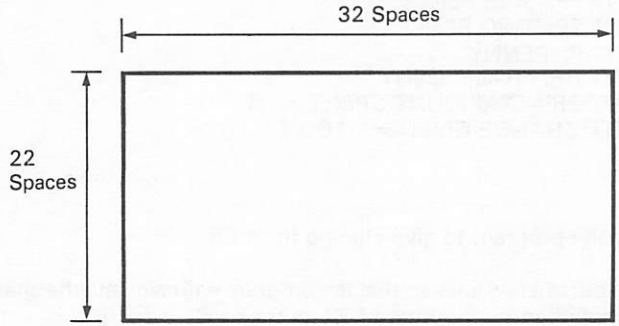
CHAPTER 4

Graphics

One of the most useful applications of a modern micro is to draw high quality graphics on the screen but the computers dealt with in this book handle graphics so differently that they have to be described separately.

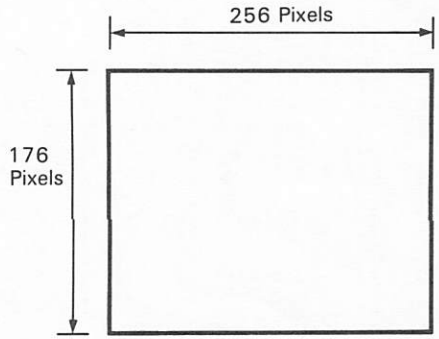
The Spectrum

The Spectrum display measures:



Each of these 704 squares (32 x 22) is subdivided into 64 smaller squares or pixels.

This gives us the high resolution display which measures:



So, in high resolution graphics we have 45056 possible pixel positions.

The computer has four graphics commands: PRINT AT, PLOT, DRAW and CIRCLE. The PRINT AT command uses low resolution graphics - the 704 squares. PLOT, DRAW and CIRCLE use high resolution graphics - the 45056 pixels. A short program will show each one in use.

This line:

```
10 PRINT AT 11,16;" A "
```

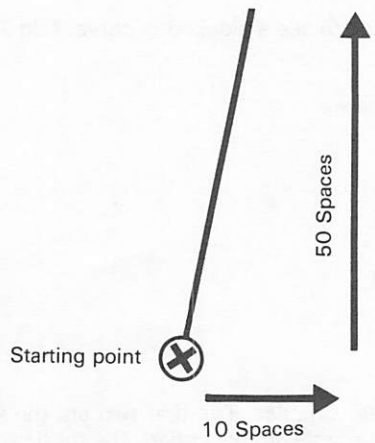
prints an A in the middle of the screen.

Now try this program using PLOT

```
10 FOR x=1 TO 100
20 FOR y=1 TO 100
30 PLOT x,y
40 NEXT x
50 NEXT y
```

It plots two lines using a series of dots on the screen; x is the horizontal co-ordinate and y the vertical co-ordinate.

The DRAW command will draw a straight line from where the previous PLOT or DRAW line ended to the position fixed by the x and y co-ordinate given with the new DRAW command. The DRAW statement is usually followed by two numbers, e.g. DRAW 10,50. The first number, 10, moves the destination point of the line to be drawn ten spaces to the right of its starting position. The second number, 50, moves the destination point 50 spaces upwards from the starting point. So DRAW 10,50 will draw this line:



Let's see if we can draw a cross in the middle of the screen using DRAW statements. PLOT 127,88 gives us the approximate centre of the screen. The North arm of the cross is produced by DRAW 0,50. To draw the South arm, we must pass through the centre. DRAW 0,-100 will take us back to the centre and then carry on for a further 50 spaces. Before drawing the Western arm we must again return to the centre. We do this by DRAW 0,50. The West arm is produced by DRAW 50,0 and the East arm by DRAW - 100,0

Here is the program:

```
10 PLOT 127,80
20 DRAW 0,50
30 DRAW 0,-100
40 DRAW 0,50
50 DRAW 50,0
60 DRAW -100,0
```

Try drawing a star and then your initials using DRAW statements.

The DRAW statement can also use a third number which makes the line curve through the number of radians indicated in the third number. Try:

```
10 PLOT 10,10: DRAW 100,110: DRAW 50,50
```

As you can see, part of a circle is drawn. If the third number is positive, the line turns to the left and, if negative, it turns to the right.

Now try a DRAW command to make an attractive pattern, using '-1'

```
10 FOR I=1 TO 175 STEP 5
20 PLOT 0,I
30 DRAW 256-I,-I
40 NEXT I
```

Try missing out the STEP instruction to see a blocked-in curve. Add 35 INK 1 to give this a colour.

Now try this multi-coloured program:

```
10 FOR I=0 TO 175
20 PLOT 0,0
30 DRAW INK RND*7,255,I
40 NEXT I
50 FOR t=175 TO 0 STEP -1
60 PLOT 0,0
70 DRAW INK RND*7,255,t
80 NEXT t
```

The CIRCLE command needs three variables. The first two are the x and y co-ordinates which fix the centre of the circle on the screen. The third variable is the

radius of the circle. This program will draw a circle in the middle of the screen.

```
10 CIRCLE 128,80,20
```

Now try this program to draw another pattern:

```
10 FOR r=1 TO 60
20 INK INT(r/10)
30 CIRCLE 128,80,r
40 NEXT r
```

We can draw a square in the middle of the screen with the next program:

```
10 PLOT 90,60
20 DRAW 0,50
30 DRAW 50,0
40 DRAW 0,-50
50 DRAW -50,0
```

An alternative way to draw a square is to use a FOR/NEXT routine.

```
10 FOR a=10 TO 20
20 PRINT AT 10,8;"*"
30 PRINT AT a,10;"*"
40 PRINT AT 20,a;"*"
50 PRINT AT a,20;"*"
60 NEXT a
```

Here are some more ideas for programs that draw patterns.

```
10 READ X
20 PLOT 75,40
30 DRAW 100,100,X
40 PAUSE 100
50 GO TO 5
60 DATA 525,625,4125
```

Add BORDER,PAPER and INK to make the program colourful.

Try adding different numbers in the DATA line to see if they will make a pattern.

Try this one next:

```
10 READ a
20 PLOT 60,30
30 DRAW 120,120,a*PI
40 CLS
50 GO TO 10
60 DATA 85,257,259,285,291,765,5211,21785
```

Here is a program that uses OVER 1. The OVER command is always written as OVER 1 for **on** and OVER 0 for **off**. OVER 1 puts the number, shape or letter being printed on top of anything else already there.

```
10 BORDER 1
20 PAPER 6
30 INK 3
40 PLOT 70,45
50 DRAW OVER 1,100,90,59↑3*PI
```

Try substituting other numbers for 59. We find 5,7,9,23 and 45 give rotated geometric shapes; 37 gives a pattern similar to 59 except that it rotates in the opposite direction. Some of the patterns can be made clearer by deleting OVER 1 from line 50.

There are three more statements we can use with graphics. They are BRIGHT, FLASH and INVERSE.

BRIGHT

The statement BRIGHT gives extra brightness to the colour being used. BRIGHT 1 makes the colour brighter and BRIGHT 0 makes the colour go back to its original brightness. Try this program to see it in use:

```
10 BORDER 1
20 PAPER 6
30 PRINT AT 10,13; BRIGHT 1;" Hello"
```

If we add BRIGHT 1 to each line the display does look brighter.

FLASH

The statement FLASH will make the colour flash on and off. Add FLASH 1 to the program as line 40. RUN the program to see FLASH at work. LIST the program when you are tired of the flashing display and delete line 40. Change line 10 to read: 10 BORDER 1 : FLASH 1 Run the changed program.

To see how we can use these ideas, try this program, and put your own name in line 20.

```
1 BORDER 1
2 PAPER 0
20 FOR n=1 TO 6; PRINT AT 10,10; BRIGHT 1; INK n; "JON TAYLOR"
PAUSE 50; FLASH 1: NEXT n
```

Another idea for highlighting a name uses changing colours:

```
5 BORDER 6
10 PAPER 1
25 FOR t=1 TO 10
30 PRINT AT 10,12; INK 1;"Richard": PAUSE 30
35 PRINT AT 10,12; INK 6; "Richard":PAUSE 30
40 PRINT AT 10,12; INK 7;"Richard": PAUSE 30
50 NEXT t
```

INVERSE

is very useful for highlighting particular words, number or shapes.

Try this program first:

The statement INVERSE 1 will exchange INK and PAPER colours. Try putting it at the beginning of line 30.

```
10 PRINT "This is normal.
20 PRINT INVERSE 1;"This is INVERSE.
30 PRINT "This is normal again.
```

Now add INVERSE 0 to line 20

```
10 PRINT :This is normal.
20 PRINT INVERSE 1;"This is INVERSE,"; INVERSE 0;"Isn't it nice?"
30 PRINT "This is normal again.
```

It negates INVERSE 1 and so "Isn't it nice?" appears in normal print.

We can use blocks of colour on the Spectrum. This program covers the screen in multi-coloured lines:

```
10 FOR c= 1 TO 20
20 PAPER INT(RND*7)
30 PRINT AT c,3;"
40 NEXT c
```

Leave 30 spaces between the inverted commas in line 30.

Add these lines to your program:

```
11 FOR c=1 TO 28
25 NEXT c
```

Run the changed program and then change 30 to 30 PRINT ATc,r;" "

Remember to leave the 30 spaces between the "" marks.

Let us suppose that we want to colour in just part of the screen. Try this program.

```
10 PAPER 2 : FOR b=16 TO 21
20 PRINT AT b,0 ;"    "
30 NEXT b
```

This prints a coloured rectangle at the bottom left hand corner line of the screen.

There will be times when we will want to put another colour over one already there. We use a POKE number to help us to do this.

Add these lines to your program :

```
1 BORDER 0
2 PAPER 0
3 CLS
```

These three lines set up the screen so that the colours can be seen.

Now type:

```
40 PAPER 6 : POKE 23694,12
50 FOR y = 10 TO 18
60 PRINT AT y,5;"    "
70 NEXT y
```

Run the program and then add the lines:

```
80 PAPER 4:POKE 23694,12
90 FOR w=8 TO 16
100 PRINT AT w,4;"    "
110 NEXT w
```

The POKE number 23694,12 allows us to put one colour over another on the screen.

We can give the impression of 3D pictures by drawing patterns almost on top of each other but slightly to one side and slightly smaller or larger. Try each of these programs and see if you can work out why they work.

```
10 LET s=20
20 FOR n= 1 TO 55
30 CIRCLE s,80,n
40 LET s=s+3
50 NEXT n
```

```
10 LET s=180
20 FOR n= 55 TO 1 STEP -1
30 CIRLCE s,80,n
40 LET s=s-3
50 NEXT n
```

Both cones look as if they are drawn in 3D. Now try these two programs:

```
10 LET s=80
STEP-1
20 FOR n=50 TO
30 CIRCLE s,80,n
40 LET s=s-3
50 NEXT n
```

```
10 LET s=1
20 LET n=1
30 CIRCLE 120,n,s
40 LET s=s+5
50 LET n=n+10
60 IF n>120 THEN
   PRINT "A nice
   pattern" : STOP
70 GO TO 30
```

Can you write a program to draw a cone pointing downwards from the top of the TV screen?

Now let us try to draw a square in 3D. Try this program:

```
10 LET p=1
20 LET q=1
30 LET a=0
40 LET b=50
50 LET c=-50
60 PLOT p,q
70 DRAW a,b
80 DRAW b,a
90 DRAW a,b
100 DRAW c,a
110 LET p=p+10
120 LET q=q+10
130 IF p>100 THEN STOP
140 GO TO 60
```

Run the program and then change lines 110 and 120 to:

```
110 LET p=p+15
120 LET q=q+5
```

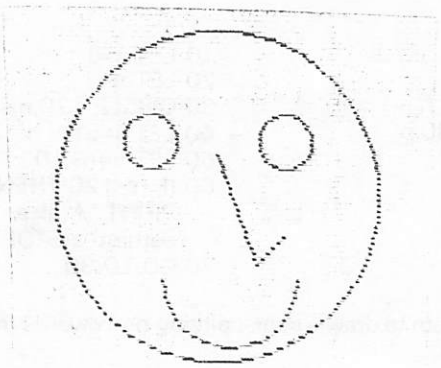
The squares are, of course, all the same size. To give a 3D effect, change line 130 to

```
130 IF p> 160 THEN STOP
```

and add lines 132 and 133 which will make the squares grow bigger.

```
132 LET b=b+5
133 LET c=c-5
```

Now let us use the computer as a sketch pad. First a funny face:



Add INK colours to make the face more amusing.

```
10 CIRCLE 120,70,60
20 CIRCLE 95,90,10: CIRCLE 145,90,10
30 PLOT 120,85: DRAW 15,-40: DRAW 10,10
40 PLOT 100,40: DRAW 50,0,PI
```

Try drawing happy and sad faces and the faces of your friends.

We can also use the computer as a plotting machine. This program will start in the middle of the screen and let you draw pictures as sequences of "+" signs.

```
10 LET x=10: LET y=15
20 PRINT AT x,y;"+"
30 PAUSE 20
31 LET d$=INKEY$
32 IF d$="5" THEN LET y=y-1
33 IF d$="8" THEN LET y=y+1
34 IF d$="6" THEN LET x=x+1
35 IF d$="7" THEN LET x=x-1
40 GO TO 20
```

You use the 5,8,6 and 7 keys to move to the next position using the addition sign. If you want to restart another drawing at a different place without these being a trail of + signs, you need to put extra lines in the program which will leave spaces and allow you to move about the screen like this:

```
10 LET x=10: LET y=15
20 PRINT AT x,y;"+"
30 PAUSE 20
31 LET d$=INKEY$
```

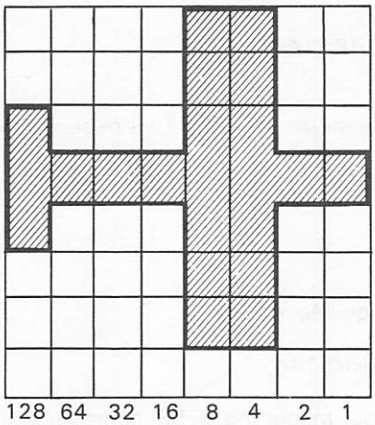
```
32 IF d$="5" THEN LET y=y-1
33 IF d$="8" THEN LET y=y+1
34 IF d$="6" THEN LET x=x+1
35 IF d$="7" THEN LET x=x+1
36 IF d$="r" THEN PRINT AT x,y;" ": LET y=y-1
37 IF d$="t" THEN PRINT AT x,y;" ": LET y=y+1
38 IF d$="y" THEN PRINT AT x,y;" ": LET x=x+1
39 IF d$="u" THEN PRINT AT x,y;" ": LET x=x- 1
40 GO TO 20
```

User Defined Graphics

To display your own shapes, such as Space Invaders, you have to use User Defined Graphics - shortened to UDGs.

There are three ways, at least, of putting UDGs on the screen when using the Spectrum. We will use a program based upon READ and DATA statements.

The first task is to draw our picture on an 8 x 8 grid (8 because the Spectrum has an 8 bit processor). Shade in, on this grid, the squares that you need for your shape.



We have drawn a small aeroplane. Now look at the numbers underneath the 8 x 8 square. They are being used to number each square. If you add the numbers given to each shaded square we will get a total which can only describe one shape.

The top line has squares with values 8 and 4 shaded, so it has a total of 12. Lower down the aircraft fuselage has squares 1,2,4,8,16,32,64 and 128 shaded and so gets a total of 255. These totals go into the DATA line of the program.

The Sinclair manual (pages 93 to 94) uses the BIN method to store this information but we think that our method is easier to understand. Your program could now be:

```
10 FOR g = 0 TO 7
20 READ d
30 POKE USR "a" + g,d
40 NEXT g
50 DATA 12,12,140,255,140,12,12,0
```

Lines 10 and 40 make sure the computer fills an 8 x 8 square. Line 20 reads the numbers we put into the DATA line from our drawing. Line 30 labels and stores the drawing as a.

To make use of the program we need to go into Graphics mode. Do this by pressing CAPS/SHIFT and 9. The cursor will become a letter G showing that the computer is in Graphics mode. Press "a" to see the little aeroplane at the bottam left hand corner of the screen. Press 9 again to get back into L mode. We need to change the program to make any real use of our drawing. Add line 70 PRINT CHR$ 144;" " (CHR$ 144 is the character code for letter a. Look at your Sinclair manual, page 186, to see this character code). RUN the program to see our aeroplane at the top of the screen.

Now add two new lines, and change line 70:

```
65 FOR a = 1TO 10
70 PRINT TAB(a);CHR$144;" "
75 NEXT a
```

Now you can see your aeroplane fly! The TAB moves it to the next position.

Let's try a nested loop:

```
66 FOR f = 1 TO 2
73 NEXT f
```

You'll also need to change line 70 to

```
70 PRINT AT a,f; CHR$144; " "
```

You can change the layout by putting STEP increments in lines 65 and 66.

Now let's make a space invader move across the screen.

In the following program, lines 10 to 50 set up the space invader image. Lines 70 and 71 set the starting position for the image to be printed in line 80. Line 86 prints a blank space in place of the space invader image. Line 85 makes the computer, when it reaches the right hand edge of the screen, return to the left hand side and begin again.

```
10 FOR I=0 TO 7
20 READ i
30 POKE USR "s"+I,i
40 NEXT I
50 DATA 60,36,255,189,189,35,66,129
60 CLS
70 LET X=10
71 LET y=1
80 PRINT AT x,y;CHR$ 162
81 PAUSE 20
82 LET s=x
83 LET p=y
84 LET y=y+1
85 IF y>28 THEN LET y=2 AND x=10
86PRINT AT s,p;""
90 GO TO 80
```

By altering line 85 you will be able to make the space invader restart his journey from a different place.

Two useful statements in games are SCREEN$ and ATTR (which stands for ATTRibute). Let us deal with each in turn.

SCREEN$ will examine the screen location we give it. We can use this to advantage in games. Try this program to see SCREEN$ in action:

```
10 PRINT AT 10,10;"a"
20 IF SCREEN$ (10,10)="a" THEN PRINT AT 20,20;"yyy"
```

Line 10 tells the computer to print a letter a near the middle of the screen. Line 20 tells the computer to look at location 10,10 and, if it is a letter a, to print at a different location three letter y's. Unfortunately, SCREEN$ will not recognise UDGs but we use the ATTR statement to do this. The ATTR statement examines the ATTRibutes of the location, i.e. colours, brightness and whether it is flashing or not. So,we can now distinguish a coloured UDG from its background.

```
10 FOR c=1 TO 6
20 INK c: PRINT AT 12,15;"-"
30 PAUSE 50
40 PRINT ATTR (12,15)
50 NEXT c
```

Let's make up a little game to show ATTRibute in use. You just have to avoid the black squares. The ATTRibute for a black square is 56 and this is used in line 130. Try constructing a maze using the ATTRibute idea to stop you going through walls or obstructions.

```
10 FOR o=1 TO 30
14 LET x=INT (RND*20)
15 LET y=INT (RND*30)
```

```
20 INK 0: PRINT AT x,y;"-"
30 NEXT o
50 LET x=1: LETY y=1
60 INK 1: PRINT AT x,y;"+"
70 PAUSE 20
80 LET d$=INKEY$
90 IF d$="5" THEN LET y=y-1
100 IF d$="8" THEN LET y=y+1
110 IF d$="6" THEN LET x=x+1
120 IF d$="7" THEN LET x=x-1
130 IF ATTR (x,y)=56 THEN PRINT AT x,y;"BANG"
140 GO TO 60
```

Graphics on the BBC

MODE, MOVE and DRAW

The screen used with the BBC computer can be arranged in a variety of different ways, according to which of several different modes you have chosen. When you first switch on the machine, the computer is set to MODE 7, the Teletext mode. The table below shows the modes available and their text screen sizes for the BBC Model A and B.

Model A	Model B	Text Screen Size	Colours
	0	80 * 32	2
	1	40 * 32	4
	2	20 * 32	16
	3	80 * 25	2
4	4	40 * 32	2
5	5	20 * 32	4
6	6	40 * 25	2
7	7	40 * 25	Teletext

The most commonly used graphics modes are MODE 2 and MODE 5. If you have a Model B computer you could type MODE 2 instead of MODE 5 for all the programs in this section. You will then notice that the lines drawn are much finer (higher resolution) than in MODE 5. MODES 0 and 3 are useful for word processing. MODES 4 and 6 have only a two colour display whilst MODE 6 doesn't support graphics.

In MODE 5 the screen can be used by the graphics statements MOVE, DRAW and PLOT. Now the display becomes 1279 "boxes" across and 1023 up. The statement MODE 5 clears the screen, puts the drawing cursor at the bottom left corner and labels it 0,0. The top right is 1279,1023.

This program should draw a diagonal line between these two corners:

```
10 MODE 5
20 DRAW 1279,1023
```

If we wanted to draw the other diagonal on its own, then we would need this program:

```
10 MODE 5
20 MOVE 0,1023
30 DRAW 1279,0
```

So MOVE tells the computer to put the drawing cursor at a point without drawing, while DRAW will join the points.

In MODE 5 there are four colours available:

0	black
1	red
2	yellow
3	white

But these can be changed by using the VDU statement. So VDU 19,0,4,0,0,0 changes colour 0, black, to colour 4, blue, so we see a blue background to the screen.

```
10 MODE5
20 VDU 19,0,4,0,0,0
30 FOR I=0 TO 1023 STEP 25
40 MOVE I+128,0
50 DRAW 1151,I
60 DRAW 1151-I,1023
70 DRAW 128,1023-I
80 DRAW I+128,0
90 NEXT I
```

The above program uses the MOVE and DRAW statements to construct a curve stitch. When you have tried it, put in this line:

```
25 GCOL,1
```

This statement tells the computer that the Graphics COLour is to be colour 1 - red.

Try these two programs which also draw a string type pattern.

```
10 MODE5
20 VDU19,3,4,0,0,0
30 SHIFT=140
40 FOR I=0TO500STEP25
```

```
50 MOVE SHIFT,500-I
60 DRAW I+SHIFT,0
70 MOVE 500+I+SHIFT,0
80 DRAW 1000+SHIFT,I
90 MOVE 1000+SHIFT,500+I
100 DRAW 1000+SHIFT-I,1000
110 MOVE 500+SHIFT-I,1000
120 DRAW 140,1000-I
130 NEXT I
```

```
10 MODE5
20 VDU19,1,4,0,0,0
30 FOR I=0TO1023STEP25
40 MOVE I+128,0
50 DRAW 1151-I,1023
60 MOVE 128,1023-I
70 DRAW 1151,I
80 NEXT I
```

When you have tried them, try altering the VDU 19 statement to other values. You could try VDU 19,3,9,0,0,0. The table below will give you some help as to what to expect:

FOREGROUND	BACKGROUND	COLOUR
0	128	black
1	129	red
2	130	green
3	131	yellow
4	132	blue
5	133	magenta
6	134	cyan
7	135	white
8	136	black-white flash
9	137	red-cyan flash
10	138	green-magenta flash
11	139	yellow-blue flash
12	140	blue-yellow flash
13	141	magenta-green flash
14	142	red-cyan flash
15	143	white-black flash

In addition to the VDU 19 statement, we can alter the colour of text on the screen by using the statement COLOUR. The table above can also be used to design the way you would like your words to appear. So, if you would like the word FIRE!! to appear in red letters on a white background you would type:

```
COLOUR 131:COLOUR 1:PRINT"FIRE"
```

This program uses INPUT so that we can put our own values into the program and see where the points are on the screen. You will also notice that the program can not stop until we press the ESCAPE key. This is because REPEAT...UNTIL FALSE means "forever"!

```
10 MODE5:COLOUR133:COLOUR2
20 REPEAT
30 PRINT TAB(0,0)STRING$(40," ")
40 INPUT TAB(0,0) " ACROSS ";ACROSS
50 INPUT TAB(3,1) " UP ";UP
60 IF ACROSS>1279 OR UP>950 THEN 30
70 DRAW ACROSS,UP
80 UNTIL FALSE
```

This "Art program uses another BBC statement - PROC (Procedure). This is a way of breaking down a program into smaller parts and then calling each part by name. So PROCX tells the computer to go to DEFPROCX and then return when it meets ENDPROC.

```
10 MODE5:VDU19,0,5,0,0,0
20 PROCX:PROCY
30 MOVEX,Y
40 REPEAT
45 GCOL0 RND(3)
50 PROCX:DRAWX,Y
55 FORI=1TO100
60 PROCY:DRAW X,Y
70 UNTIL FALSE
1000 DEFPROCX
1010 X=RND(1279)
1020 ENDPROC
2000 DEFPROCY
2010 Y=RND(1023)
2020 ENDPROC
```

PLOT

The PLOT statement in BBC BASIC can do many things. After PLOT come three numbers. The first tells the computer which kind of plot you want and the other two numbers show which points on the screen are required as in the MOVE and DRAW statements. Try:

```
MODE 5:PLOT 0,100,100
```

Nothing should happen! This is the same as the statement MOVE100,100. Now Type:

```
PLOT 1,200,500
```

Now you should see a line. PLOT 1 means MOVE but ADD the numbers to the last co-ordinates. So you should now be at 700,600. To check this, try typing:

```
PLOT 1,-500,-500
```

PLOT 2 also ADDS but the lines are drawn in the "opposite" colour.

PLOT 3 draws in the background colour.

PLOT 4,5,6 and 7 do the same job as 0,1,2 and 3 but do not ADD to the co-ordinates, but move directly to the points given in the PLOT statement.

Adding 16 to the first number after the PLOT statement draws dotted lines. Adding 80 to the first number after the PLOT statement draws triangles. As an example try this doodling program. It uses a PLOT 5 to draw a line in red (GCOLO,1) but uses the arrow cursor keys (INKEY) to get its direction. You could try altering SPEED and GCOLO,1. The VDU 5 actually gets rid of the cursor. You will have to ESCAPE from this program!

```
10 MODE5:VDU:GCOL0 1
20 SPEED=5:LOW=0:HIGH=1023:SIDE=1279
40 X=640:Y=512:MOVEX,Y
60 REPEAT
70 X1=INKEY(-26):X2=INKEY(-122):REM LEFT RIGHT
80 Y1=INKEY(-42):Y2=INKEY(-58):REM DOWN UP
90 X=X+(SPEED*(X1-X2)):Y=Y+(SPEED*(Y1-Y2))
100 IF X>=SIDE THEN X=SIDE
110 IF X< LOW THEN X=LOW
120 IF Y>=HIGH THEN Y=HIGH
130 IF Y< LOW THEN Y=LOW
140 PLOT 5,X,Y
150 UNTIL FALSE
```

Another example is this simple program which draws 100 coloured triangles on the screen using PLOT 85.

```
  5 MODE5
 10 FORI=1TO100
 20 GCOL0 RND(3)
 30 PLOT 85,RND(1279),RND(1023)
 40 NEXT
```

The last program drew triangles in a random fashion all over the screen, the computer selecting its own random places in the statements RND(1279) for horizontal point and RND(1023) for the vertical point. Now we will try to draw triangles in a regular way by joining the centre of the screen to the outside edges, first the bottom, then the right side, the top and finally the left side.

```
10 INPUT"HOW WIDE ";BAND
20 MODE 5:C=1
```

```
30 MOVE 0,0
40 FOR I=0 TO 1280 STEP-BAND
50 PROCMOVE
60 PLOT 85,I,0
70 NEXT
80 FOR I=0 TO 1024 STEP-BAND
90 PROCMOVE
100 PLOT 85,1280,I
110 NEXT
120 FOR I=1280 TO 0 STEP-BAND
130 PROCMOVE
140 PLOT 85,I,1024
150 NEXT
160 FOR I=1024 TO 0 STEP-BAND
170 PROCMOVE
180 PLOT 85,0,I
190 NEXT
200 RUN
210 DEFPROCMOVE
220 MOVE 640,512
230 GCOL0,C
240 C=C+1:IF C=4 THEN C=0
250 ENDPROC
```

When you run this program, enter 100 when asked "HOW WIDE?" The computer will then draw triangles from the centre to the edge of the screen so that their bases are the width you specified. Notice the four separate FOR...NEXT loops in the program for each of the sides of the screen. The variable C is used as a counter to keep track of the colours that are printed. These will change each time (lines 230,240) and will cycle through the four colours available in MODE 5. Try adding a VDU 19 statement at line 25 to change the colours used (VDU 19,0,4,0,0,0). Also vary your reply to the HOW WIDE question. Try very small numbers.

Drawing Circles

There is no statement on the BBC computer to draw circles directly. However, there are many statements which have room in them for later expansion - the 0,0,0 at the end of the VDU 19 for example.

So, we have to resort to a mathematical way of using PLOT to draw the circle. If you imagine a circle cut up into many isosceles triangles, this is how the computer could draw it.

Type this program and then run it. You should see a reasonable circle:

```
10 MODE 5
20 MOVE 640,512
50 GCOL0,1
120 FOR I=0 TO 2*PI STEP PI/10
```

```
130 MOVE 640,512
150 C1=640+400*COS(I):C=512+400*SIN(I)
160 PLOT 85,C1,C2
170 NEXT I
```

Now try altering the 10 in line 120 to other values. The smaller it becomes, the more the circle can be seen to be a number of joined-up triangles, like a neatly-sliced cake. Type in the extra lines so that your program looks like this one. You should now have a program which will display fractions in piechart form:

```
10 MODE 5
20 MOVE 640,512
30 PRINT
40 VDU19,0,4,0,0,0
50 GCOL0 1
60 COLOUR2:PRINTTAB(1,1);:INPUT"NUMERATOR ? "N
70 PRINTTAB (1,1)STRING$(20," ")
80 PRINTTAB(1,1);:INPUT"DENOMINATOR ? "D
90 PRINTTAB(1,1)STRING$(20, " ")
100 PRINTTAB(0,1)"FRACTION ";N;"/";D;"   ":FR=N/D
110 PRINTTAB(0,2)"DECIMAL   ";FR
120 FOR I=0 TO 2*PI STEP PI/10
130 MOVE 640,512
140 IF I>=(FR*2*(PI+0.004*PI)) THEN GCOL0,3
150 C1=640+400*COS(I):C2=512+400*SIN(I)
160 PLOT 85,C1,C2
170 NEXT I
180 PRINTTAB(0,29)"ANOTHER ?"
190 A$=GET$
200 IF A$="Y" THEN RUN ELSE END
```

User Defined Graphics

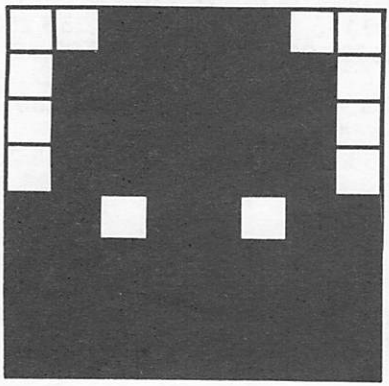
When you play Space Invaders or Pacman, have you ever wondered how the programmers have managed to get the characters on the screen when there are no graphics keys on the BBC machines?

The answer is that they design each shape, turn it into a pattern of numbers, and then program the computer to draw it. Let's try drawing a simple Jack-in-the-box. If we put these shapes together:

224 225 226 227 228 229 230

then we get:

Each of the shapes was drawn first on an 8 x 8 grid. The squares were shaded in to make the pattern we wanted. Since each of the columns of squares has a value, we can total up each row and convert our pattern into a "digital" picture that the computer can work on. Look at the grid below. The shape of the top of the Jack's head has been filled in.



So, working from the top of the grid downwards, we have row values of:

60,126,126,126,255,219,255 and 255.

Try programming the computer to draw this shape using this program:

```
20 MODE 5
170 VDU23,225,60,126,126,126,255,219,255,255
460 COLOUR1:PRINT TAB(9,11);CHR$225
```

When you RUN it you should see a red face half way down the screen.

The rest of the program is listed below. The idea is to produce a simple reaction timer. A box is drawn and after a random delay the box opens and out springs the Jack. You then have to press any key in the shortest possible time. Your time is displayed before the next "spring".

Here is the full program for the reaction timer. Notice that it uses procedures. The main program consists of lines 10 to 120, each procedure being called when needed.

```
10 REPEAT                                   Start of main program loop.
20 MODE5                                    Graphics mode.
30 PROCCHARACTERS                           To define the shapes.
40 PROCBOX                                  To draw the closed box.
50 PROCDELAY(RND(10000))                    Random pause.
60 PROCLID                                  Open the box.
70 PROCNOISE                                Make a springy noise.
80 PROCJACK                                 Draw the jack.
90 PROCTIMER                                Calculate the reaction time.
100 PROCDELAY(3000)                         Pause
110 UNTIL FALSE                             End of loop.
120 END
130 :
140 DEFPROCCHARACTERS
150 VDU23;8202;0;0;0:PRINT                  Turn off the cursor.
160 VDU23,224,0,0,0,40,171,126,126,126      Define the crown.
170 VDU23,225,60,126,126,126,255,255,219,255,255  Define top of head.
180 VDU23,226,102,102,126,126,90,102,60,60  Define bottom of
                                            head.
190 VDU23,227,24,24,126,255,255,255,126,60  Define body.
200 VDU23,228,32,60,4,60,32,60,4,60         Define spring.
210 VDU23,229,255,255,255,255,255,255,255,255  Define one block.
220 VDU23,230,128,128,128,128,128,128,128,128  Define lid.
230 ENDPROC
240 :
250 DEFPROCBOX
260 PRINTTAB(3,0)"REACTION TIMER"           Titles
270 COLOUR2:FORI=15TO19:PRINTTAB(8,I); CHR$229;CHR$229;
    CHR$229;:NEXT                           Draw 5 x 3 blocks.
280 ENDPROC
290 :
300 DEFPROCLID
310 FORI=10TO14:PRINTTAB(8,I);CHR$230:NEXT  Draw 4 lid sections.
320 ENDPROC
330 :
340 DEFPROCNOISE
350 *FX15                                   Empty sound memory.
360 ENVELOPE1,1,0,0,88,0,0,50,117,-1,-6,0,126,92  Define shape of
                                            noise.
370 SOUND 1,1,10,20                         Make the sound.
380 ENDPROC
390 :
400 DEFPROCDELAY(T)                         Delay from main loop.
410 FORI=1TOT:NEXT
420 ENDPROC
430 :
440 DEFPROCJACK
450 COLOUR2:PRINTTAB(9,10);CHR$(224)        Draw crown.
460 COLOUR1:PRINTTAB(9,11);CHR$(225)        Draw top of head.
```

```
470 PRINTTAB(9,12);CHR$(226)          Draw bottom of head.
480 PRINTTAB(9,13);CHR$(227)                Draw body.
490 COLOUR3:PRINTTAB(9,14);CHR$(228)        Draw spring.
500 ENDPROC
510 :
520 DEFPROCTIMER
530 *FX15,1                           Empty keyboard memory.
540  TIME=0:A$=GET$:COLOUR3:PRINTTAB(4,28);
     TIME/100; " SECONDS"                      Display time.
550 ENDPROC
```

Teletext Graphics

Drawing in Teletext mode (MODE 7) involves using CHR$ followed by a number which tells the computer which effect you want for that line only. The list of effects is shown below:

CODE	EFFECT	TRY TYPING
129	red letters	PRINT CHR$129"FIRE"
130	green letters	PRINT CHR$130"GRASS"
131	yellow letters	PRINT CHR$131"BANANAS"
132	blue letters	PRINT CHR$132"SEA"
133	magenta letters	PRINT CHR$133"PURPLE?"
134	cyan letters	PRINT CHR$134"SKY BLUE"
135	white letters	PRINT CHR$135"SNOW"
136	FLASH	PRINT CHR$136"BLINKING"
137	STOP FLASH	PRINT CHR$135"FLASH"CHR$137" GORDON"
145	red graphics	PRINT CHR$145"aaaaaaaaaa"
146	green graphics	PRINT CHR$146"bbbbbbbbbb"
147	yellow graphics	PRINT CHR$147"cccccccccc"
148	blue graphics	PRINT CHR$148"dddddddddd"
149	magenta graphics	PRINT CHR$149"eeeeeeeeee"
150	cyan graphics	PRINT CHR$150"ffffffffff"
151	white graphics	PRINT CHR$151"gggggggggg"
154	SEPARATED graphics	PRINT CHR$154;CHR$151"hhhhhhh"
157	CHANGE BACKGROUND	PRINT CHR$157;CHR$129"FIRE"

Each of the characters inside the quotation marks is changed into the corresponding graphics symbol. These Teletext graphics are different from user defined characters because the grid they use is 2 x 3 "pixels" wide. In addition there is no corresponding character for a whole block of colour. Instead we have to use CHR$255 (as in user defined graphics).

Before typing in the Teletext picture program, try some of these simpler programs:

Moving Car

```
10 MODE 7
20 FORI=0TO35:PRINTTAB(I,10)CHR$145".?o-"
25 FORP=1TO100:NEXT
30 NEXT
```

Teletext Colours

```
10 MODE 7
20 SENTENCE$="  THESE ARE TELETEXT COLOURS"
30 FOR BACKGROUND=136TO129 STEP-1
40 FOR SHADE=129TO136:PRINTCHR$BACKGROUND;CHR$157;CHR$DE;
   SHADE; SENTENCE$
50 FOR PAUSE=1TO1000:NEXT PAUSE
60 NEXT SHADE
70 NEXT BACKGROUND
```

Teletext Titles

```
10 MODE 7
20 PRINTCHR$150" <,,4<,,4<,,4"
30 PRINTCHR$148CHR$157CHR$150"5b(55b(55j"CHR$255"5"
50 PRINTCHR$150" -,,%-,,%-,,%"
```

To change the background colour we need to specify three CHR$ codes. First give the colour code, and then the background code. For example, try:

```
PRINT CHR$ 130;CHR$157
```

This gives a green background. If you do not follow this with another colour change, you will not see what has been printed! Try:

```
PRINT CHR$ 130;CHR$ 157"GRASS IS GREEN".
```

To rectify this, type:

```
PRINTCHR$130;CHR$157;CHR$135"GRASS IS GREEN".
```

And now for the Teletext picture. We shall draw a scene with Stars, a Fence, a Streetlight and a House.

Again, the program will use procedures so the main program will be a short list of the procedures to be called (Lines 20 to 60).

```
10 MODE 7                                   Set teletext mode
20 PROCSTARS
30 PROCFENCE
40 PROCLIGHT
```

```
50 PROCLAMP
60 PROCHOUSE
70 END
80 :
90 DEFPROCSTARS
100 REPEAT
110 PRINTTAB(RND(40),RND(10))CHR$136"."      Flashing stars (136)
120 PRINTTAB(RND(40),RND(10))"."                    Steady stars.
130 C=C+1
140 UNTILC=10                                        10 of each type.
150 ENDPROC
160 :
170 DEFPROCFENCE
180 FORI=16TO23
190 PRINTTAB(0,1)CHR$146"/j/j/j/j/j/j/j/j/j"   Green graphics(146)
200 NEXT
210 ENDPROC
220 :
230 DEFPROCLIGHT
240 PRINTTAB(22,13)CHR$150"7£5"                  Cyan graphics (150)
250 PRINTTAB(24,14)CHR$147"!"                   Yellow graphics (147)
260 ENDPROC
270 :
280 DEFPROCLAMP
290 FORI=14TO23
300 PRINTTAB(22,I)CHR$150"5"                     Cyan graphics (150)
310 NEXT
320 ENDPROC
330 :
340 DEFPROCHOUSE
350 FORI=11TO13:REM ROOF
360 PRINTTAB(26,I)CHR$154CHR$148    Separated blue graphics
    STRING$(12,CHR$255)                    (154 148)
370 NEXT
380 FOR I=14TO23:REM WALLS
390 PRINTTAB(27,I)CHR$129CHR$157;    Change background to red
                                                (129 157)
400 IFI=15ORI=16ORI=19ORI=20THEN
    PRINTCHR$151"**          **":REM  WINDOWS
410 IFI>20THENPRINT"    ";CHR$151;CHR$148 White graphics (151)
    CHR$255;CHR$255:REM DOOR                  Blue graphics (148)
420 NEXT I
430 ENDPROC
```

VIC 20 Graphics

There are three ways of producing graphics characters on the VIC. They are:

1. By using the keyboard symbols.
2. By using the CHR$ function.
3. By employing user-defined characters.

In all the programs for the VIC that follow, the keyboard graphic symbols have been replaced in the listings by the actual keys to press. So, although the symbol for clear screen is an inverted heart, any listings will show [CLR]. When using this symbol, the SHIFT key should also be pressed, since CLR is the top character on this key. Where this is not clear, the listing might also show that a SHIFT, CTRL or COMMODORE key should be pressed at the same time. For example

[SHIFT+G] means hold down SHIFT and press G.

[COMMODORE+G] means hold down the COMMODORE key and press G.

[CTRL+G] means hold down the CTRL key and press G.

Occasionally the cursor keys need to be pressed more than once when building up a string. Thus:

[13 CRSR DOWN] means press the cursor down key thirteen times.

Keyboard Graphics

VIC users are well supplied with a wide variety of characters directly printable from the keyboard. The VIC has inherited the same range of "graphical" characters as the first CBM PET machines. As these were considered to be business machines, the construction of graphs and charts is fairly simple.

Try this program to draw a grid:

```
10 PRINT"[CLR]";
20 A1$="[COMMODORE+A]":A2$="[COMMODORE+R]":A3$="
   [COMMODORE+S]"
30 PRINTA1$;:FORI=1TO19:PRINTA2$;:NEXT:PRINTA3$
40    B1$="[COMMODORE+Q]":B2$="[SHIFT++]":B3$="
   [COMMODORE+W]"
50 FORR=1TO20
60 PRINTB1$;:FORI=1TO19:PRINTB2$;:NEXT:PRINTB3$
70 NEXT
80 C1$="[COMMODORE+Z]":C2$="[COMMODORE+E]":C3$="
   [COMMODORE+X]"
90 PRINTC1$;:FORI=1TO19:PRINTC2$;:NEXT:PRINTC3$
100 GETA$:IFA$=""THEN100
```

Be careful not to miss the semicolons (print on the same line), especially the one following the CLR. Leaving it out will spoil the screen display, as the screen will clear and printing will start one line down instead of at the very top of the screen.

When you RUN this program, the pattern that is generated is the sort that TV engineers use to check television displays for alignment, but it could easily be used in a games or mathematical program.

Here is a program which makes use of vertical bar graphics. Close examination of the keyboard of the VIC will show that there are several keys which produce vertical lines. These lines are set at slightly different distances from the edge of the character block.
Altogether there are eight of these characters. Starting with the one that draws a line at the extreme left hand edge and working up to the character that draws a line at the extreme right hand edge, they are:

Commodore key + G
Shift + T
Shift + G
Shift + B
Shift + -
Shift + H
Shift + Y
Commodore key + M

The next program concerns decimals, and is a sort of ruler. When you run the program you are asked for your value. Input a decimal number between 1 and 20, say 11.3, and the computer will draw that position on a number line it has drawn.

```
10 PRINT "[CLR]"
20 S$="[20 SHIFT+0][COMMODORE KEY+G]"
30 A$="[COMMODORE KEY+G][SHIFT+T][SHIFT+G][SHIFT+B][SHIFT+-]
   [SHIFT+H][SHIFT+Y[[COMMODORE KEY+M]"
40 D$="[HOME][14 CRSR DOWNS]"
50 FORI=1TO21:C$=C$+"[SPACE]":NEXT
60 PRINTD$;"[2 CRSR DOWNS]";S$
70 X=0:X1=0:X2=0
80 INPUT"[CRSR DOWN]YOUR VALUE";X
90 IF X>20 THEN GOTO60
100 PRINTD$;C$
110 IF X=INT(X) THEN GOTO 140
120 X1=INT(X):X2=INT(10*(X-X1)+0.01):IF X2=9 THEN X2=8
130 PRINTD$:PRINT SPC(X1);MID$(A$,X2,1):GOTO60
140 PRINTD$:PRINT SPC(X);"[COMMODRE KEY+G]"
150 GOTO60
```

Line 10 clears the screen.
Line 20 defines the number line string.
Line 30 defines the pointer line string.

Line 40 defines the screen print position string.
Line 50 defines a rubbing out string (21 spaces)
Line 60 prints number line 15 lines down.
Line 70 initialises variables for numbers.
Line 80 gets input.
Line 90 throws out unacceptable inputs.
Line 100 rubs out any text 14 lines down.
Line 110 checks for whole number inputs.
Line 120 works out the integer and decimal part of input:also corrects for eight divisions.
Line 130 prints pointer at correct position on number line.
Line 140 prints whole number pointer.

It is interesting to note here that the computer is not infallible. Notice on line 120, 0.01 has to be added to the formula that works out the decimal part of the input. This is because of an inaccuracy in the 6502 processor's method of dealing with maths. Try the following:

```
PRINT INT(10*(9.9-9.0))
```

Now if you work this out, (9.9-9.0) is 0.9 and 10 times 0.9 is 9.0 so the INTeger part is 9. The computer gives 8!!!!

"Vicky" is an example of a program which uses the keyboard graphics to build up a larger picture, this time a girl's face.

Take care when typing in the program, remembering that the text between brackets shows the *actual* keys to press. Also be careful not to miss any of the characters that are sandwiched between brackets.

```
10 PRINT"[CLR][3 SPACES][CTRL+8]@@@@@"
20 PRINT"[2 SPACES]@@[CTRL+3][CTRL+9][3 SPACES] [CTRL+0]
   [CTRL+8]@@"
30 PRINT"[2 SPACES]@[CTRL+3][CTRL+9][SPACE][COMMODORE KEY+Y]
   [SPACE][COMMODORE KEY+Y][SPACE][CTRL+8]@"
40 PRINT"[2 SPACES][CTRL+9][CTRL+3][SH'FT+8][SPACE]
   [COMMODORE KEY+C][SPACE][COMMODORE KEY+C][SPACE][SHIFT+9]"
50 PRINT"[2 SPACES][CTRL+9]*[2 SPACES]V[2 SPACES]*"
60 PRINT"[3 SPACES][COMMODORE KEY+*[CTRL+9][SPACE][SHIFT+S]
   [SPACE][CTRL+0][SHIFT+#]"
70 PRINT"[4 SPACES][CTRL+9][SHIFT+M][COMMODORE KEY+@][SHIFT+N]"
80 PRINT"[SPACE][CTRL+7][CTRL+9][SHIFT+#][2 SHIFT+X][CTRL+3]
   [3 COMMODORE KEY+@][CTRL+7][2 SHIFT+X][COMMODORE KEY+*]
90 PRINT"[SPACE][CTRL+9][9 SHIFT+X["
100 A$="[CTRL+5] VICKY ":A=LEN(A$)
110 PRINT
120 FORI=1TOA:PRINTMID$(A$,I,1);:NEXT:IF R=1 THEN R=0:GOTO140
130 PRINT:PRINT"[CRSR UP] [CTRL+9]";:R=1:GOTO120
140 PRINT:PRINT"[CRSR UP] [CTRL+0]";:R=0:GOTO120
```

Line 10 clears screen and prints yellow hair.
Line 20 prints hair (still yellow), red forehead and yellow hair.
Line 30 prints hair (yellow), red eyebrows and yellow hair.
Line 40 prints red ears and eyes.
Line 50 prints earrings (still red) and nose.
Line 60 prints mouth.
Line 70 prints chin.
Line 80 prints blue dress, red neck and blue dress.
Line 90 prints dress (still blue).
Line 100 defines string and its length.
Line 110 prints a blank line.
Line 120 prints VICKY one letter at a time and checks to see if the flag R is set.
Line 130 moves the cursor back up ready to print in reverse text and sets flag R.
Line 140 moves the cursor back up to ready to print in normal text and lowers flag R.

Border/Screen Colours

Try adding this line to the Vicky program:

```
5 POKE 36879,59
```

You will notice that the border colour and screen colour are now both cyan. Try altering the 59 in line 5 to see what other effects you can achieve.

To obtain the full range of border/screen changes, RUN this program:

```
10 FORI=0TO255
20 POKE 36879,I
30 PRINT"[CLR][4 CRSR DOWNS]THIS IS THE EFFECT OF"
40 PRINT
50 PRINT"POKE 36879,";I
60 GETA$:IF A$="" THEN 60
70 NEXT
```

Graphics using the CHR$ function

In an eight bit machine like the VIC, the highest number that one memory location can hold is 255 (Hex FF). So each number from 0 to 255 can be given a symbol for printing. Look at all these symbols by typing this program directly into your computer:

```
FORI=0TO255:PRINT"CHARACTER"I;"IS ";CHR$(I):FORR=1TO500:NEXTR,I
```

You will see some strange happenings when the number for some of the computer's control codes are printed!! Some text will appear in new colours, the screen will clear and some characters will be invisible.

CHR$(148) is the code for INSERT. We can use this in a program to bump a picture across the screen. Try this short LORRY program.

```
10 PRINT"[CLR]";
20 C$="[2 SPACES][CTRL+9][CRSR DOWN][3 CRSR LEFTS][3 SPACES]
   [CRSR DOWN][3 CRSR LEFTS][CTRL+0][SHIFT+Q][CRSR RIGHT][SHIFT+Q]"
30 PRINTC$;"[HOME]";
40 FORI=1TO18
50 PRINTCHR$(148)
60 PRINTCHR$(148)
70 PRINTCHR$(148)
80 PRINT"[HOME]";
90 FORR=1TO40:NEXTR
100 NEXTI
```

It begins by defining C$ as the lorry shape and draws it at the top of the screen. Then eighteen times an INSERT character is placed at the beginning of each line on which the lorry has been drawn. The result is that the lorry moves across the screen. Try altering the delay between each "bump" by changing the 40 in line 90.

Some of the most used graphic symbols in VIC games are those representing the card suits. Here is a short program that will draw the three of hearts on the screen. You could try and alter it to display another card and then write a program around it to deal a hand of cards!

```
10 R$=CHR$(13)
20 A1$="[SHIFT+U][5 SHIFT+*][SHIFT+I]"
30 A2$="[SHIFT+-][5 SPACES][SHIFT+-]"
40 A3$="[SHIFT+J][5 SHIFT+*][SHIFT+K]"
50 FORI=1TO9:A4$=A4$+A2$+R$:NEXT
60 PRINT"[CLR]";A1$;R$;A4$;A3$
70 PRINT"[HOME][CRSR DOWN][CRSR RIGHT][CTRL+3]3"
80 PRINT"[5 CRSR RIGHT][SHIFT+S]"
90 PRINT"[2 CRSR DOWN][3 CRSR RIGHT][SHIFT+S]"
100 PRINT"[2 CRSR DOWN][CRSR RIGHT][SHIFT+S]"
110 PRINT"[5 CRSR RIGHT]3"
```

CHR$(13) is a carriage return. There is no symbol for this on the keyboard (apart from pressing RETURN of course), and so the CHR$ function has to be used. The program begins by defining R$ to be the carriage return (it is quicker and takes up less memory), A1$ to be the top of the card, A2$ to be the middle of the card and A3$ to be the bottom of the card.

Then, a FOR NEXT loop builds up the string A4$. This is nine A2$'s (card middles) each followed by a carriage return (so each middle gets printed on the next line down).

The final part of the program sends the cursor back to the top of the screen and draws the 3 and the heart symbols in red.

Here is a Commodore version the "Jack in the Box" program we developed for the BBC computer. For an explanation, just turn back a few pages, then see if you understand how it works:

```
20 PRINT"[CLR]"
150 POKE 36869,255
151 FOR CH=0TO7
153 READ A:POKE 7176+CH,A
155 NEXT CH
170 DATA 60,126,126,126,225,219,255,255
600 PRINT"[RED]A"
605 PRINT:PRINT"[RVS ON]PRESS A KEY"
610 GET A$:IF A$=""THEN 610
620 POKE 36869,240
```

When you run this you should see a red face on the screen but immediately you "PRESS A KEY" the face changes into a letter A. The POKE on line 150 allows us to redefine the shape of any of the keys. Normal letters can be printed if they are preceded by the RVS ON symbol. Line 620 puts things back to normal!

The rest of the program is listed below. The idea is to produce a simple reaction timer. A box is drawn and after a random delay the box opens and out springs the Jack. You then have to press any key in the shortest possible time. Your time is displayed before the next "spring".

When you have finished with the program type "E" and the it will End.

Here is the full program for the reaction timer. Notice that it uses subroutines. The main program consists of lines 10 to 130 where each subroutine is called when needed.

```
10D$="[RVS ON][13 CRSR DOWN]":A$="          Define positioning strings.
  [RVS ON][9 CRSR RIGHT]"
20PRINT"[CLR]"                              Clear screen.
30GOSUB 150:REM CHARACTERS                  To define the shapes.
40GOSUB 260:REM DRAW BOX                    To draw the closed box
50LET T=RND(1)*5000:GOSUB 410:REM DELAY     Random pause.
60GOSUB 310:REM DRAW LID                    Open the box.
70GOSUB 350:REM NOISE                       Make a nosie.
80GOSUB 450:REM DRAW JACK                   Draw the jack.
90GOSUB 530:REM TIMER                       Calculate the reaction time.
100LET T=1000:GOSUB 410:REM DELAY           Pause.
110:
120IF K$<>"E" THEN RUN                      Check for end of program.
130POKE 36869,240:PRINT"[CLR]":END          Restore original keys.
140REM CHARACTERS
150POKE 36869,255                           Enable redefined keys.
151FOR CH=0TO55
153READ A:POKE 7176+CH,A                    Data reading loop.
```

```
155NEXT CH
160DATA 0,0,0,40,171,126,126,126                Data for the crown.
170DATA 60,126,126,126,255,219,255,255          Data for top of head.
180DATA 102,102,126,126,90,102,60,60            Data for bottom of head.
190DATA 24,24,126,255,255,255,126,60            Data for body.
200DATA 32,60,4,60,32,60,4,60                   Data for spring.
210DATA 255,255,255,255,255,255,255,255         Data for one block.
220DATA 128,128,128,128,128,128,128,128         Data for lid.
230RETURN
240:
250REM DRAW BOX
260PRINT"[RVS ON] REACTION TIMER"               Titles.
270PRINTD$;:FORI=1TO3:PRINTA$"[GRN][RVS OFF]    Draw 5x3 blocks.
    FFF":NEXT I
280RETURN
290:
300REM DRAW LID
310PRINT"[HOME]";LEFT$(D$,12);:FORI=1TO4:       Draw 4 lid sections.
    PRINTA$;"[RVS OFF]G":NEXT I
320RETURN
330:
340REM NOISE
350POKE36878,10:POKE36876,200                   Select volume and pitch.
360FORI=1TO10:GET K$:NEXT                       Empty the keyboard
                                                memory.
370POKE36876,0:POKE36878,0                      Turn off the sound.
380RETURN
390:
400REM DELAY
410FORI=1TOT:NEXT                               Delay from main program.
420RETURN
430:
440REM DRAW JACK
450PRINT"[HOME]";LEFT$(D$,11);A$"[RVS  OFF]     Draw crown.
    [CRSR RIGHT]A"
460PRINTA$;"[RED][RVS OFF][CRSR RIGHT]B"        Draw top of head.
470PRINTA$;"[RVS OFF][CRSR RIGHT]C"             Draw bottom of head.
480PRINTA$;"[RVS OFF][CRSR RIGHT]D"             Draw body.
490PRINTA$;"[YEL][RVS OFF][CRSR RIGHT]E"        Draw spring.
500RETURN
510:
520REM TIMER
530TI$="000000"                                 Set clock to zero.
540GET K$:IF K$=""THEN 540                      Wait for key pressed.
550PRINT"[RVS ON][5 CRSR DOWN][BLK]";TI/60;     Display time.
    "SECONDS"
560RETURN
```

CHAPTER 5

Sound

Sound on the Spectrum

Sounds on the Spectrum are made with a small loudspeaker. The Spectrum statement BEEP is written with two numbers

```
BEEP X, Y
```

X is the time in X,Y seconds, Y is the musical note

Middle C is given the number 0 so

```
BEEP 1,0
```

sounds middle C for one second.

To sound all the notes of music available for half a second, type in this program:

```
10 FOR m=-60 TO 70
20 BEEP .5,m
30 NEXT m
```

You should hear a steadily rising note.

This program makes a noise like an emergency vehicle's siren:

```
10 FOR p = 1 TO 10
20 BEEP .5,10
30 BEEP .4,12
40 NEXT p
```

After RUNning the program try altering line 30 to BEEP .6,12

Now try this program for a very different noise:

```
10 FOR s = 1 TO 30
20 BEEP .1,1
30 BEEP .01,10
40 NEXT s
```

Nested loops can make some very unusual sounds:

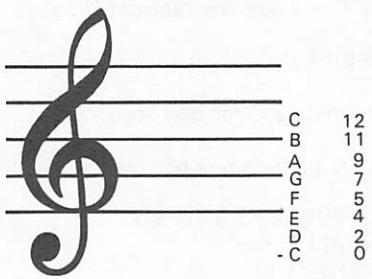
```
10 FOR I = 1 TO 10
20 n = 0 TO 20
30 BEEP .5,n
40 NEXT n
50 NEXT I
```

Making up your own tunes or making the computer play a tune is quite straightforward. First, we need to know which BEEP numbers represent which musical notes. Secondly, we need to know how to represent the length of musical notes as a BEEP number.

Middle C is given a BEEP note value of 0. As the notes get higher, the BEEP numbers increase. If a note lower than Middle C is wanted, we use negative numbers so B below Middle C would have a score of -2, whereas D above Top C is given the score of 14.

For the timing, use:

note	BEEP time value
	2
	1
	.5



Here is a simple tune for the Spectrum:

```
C  G  G  A
B  A  G  F
G  F  E  F
G  -  E  -    Repeat twice

C  C  C  G
B  A  G  F    Repeat twice

C  G  G  A
B  A  G  F
G  F  G  F
C             Restart the whole.
```

The program for this tune is:

```
 1 REM © Mrs.M. Roxburgh 1983
10 FOR m=1 TO 2
11 BEEP .5,0: BEEP .5,7 .5,7: BEEP .5,9
12 BEEP .5,10: BEEP .5,5: BEEP .5,7: BEEP .5,5
14 BEEP 1,7: BEEP 1,4
15 NEXT m
20 FOR m=1 TO 2
21 BEEP .5,12: BEEP .5,12: BEEP .5,12: BEEP .5,7
22 beep .5,10: BEEP .5,9: BEEP .5,7: BEEP .5,5
23 NEXT m
30 BEEP .5,0: BEEP .5,7: BEEP .5,7: BEEP .5,5
31 BEEP .5,10: BEEP .5,9: BEEP .5,7: BEEP .5,9
32 BEEP .5,7: BEEP .5,5: BEEP .5,7: BEEP .5,5
33 BEEP .5,0
40 GO TO 10
```

There are, then, three steps involved in changing sheet music to computer music. First, change the notes written on the staves into letters along with their time values. Second, go to Page 135 of the Sinclair manual and change these letters into numbers. Third, enter the numbers and time value with a new BEEP for each note.

Spectrum Keyboard

We can make our Spectrum into an organ. Although the sound output is not loud enough to be a really useful organ, it is good enough for us, as we are not musicians. Change the PAUSE length in line 10 to suit yourself, and then change the durations to fit the tune. You can, of course, add to the number of notes to be played by the computer by using other keys on the keyboard. Here is the initial program to start with:

```
10 PAUSE 50
11 LET m$=INKEY$
12 IF m$=" " THEN GO TO 11
13 IF m$="c" THEN BEEP .5,0
14 IF m$="d" THEN BEEP .5,2
15 IF m$="e" THEN BEEP .5,4
16 IF m$="f" THEN BEEP .5,5
17 IF m$="g" THEN BEEP .5,7
18 IF m$="a" THEN BEEP .5,9
19 IF m$="b" THEN BEEP .5,11
20 GO TO 10
```

The problem with this program is that all keys which need to be pressed are close together and not in the correct order.

We can use the bottom line of the keyboard to represent the notes C to B. Try this program to see if you like it better than the first:

```
10 PAUSE 50
11 LET m$=INKEY$
12 IF m$="" THEN GO TO 11
13 IF m$="z" THEN BEEP .5,0
14 IF m$="x" THEN BEEP .5,2
15 IF m$="c" THEN BEEP .5,4
16 IF m$="v" THEN BEEP .5,5
17 IF m$="b" THEN BEEP .5,7
18 IF m$="n" THEN BEEP .5,9
19 IF m$="m" THEN BEEP .5,11
20 GO TO 10
```

Anybody who is interested in music could assign a musical note to every key on the board. We would suggest using the keys A to L for notes from Middle C upwards. Keys Z to M could be used for notes lower than Middle C. Keys Q and P could be used for notes higher than whatever note you assign to key L.

Sound on the BBC computers

The designers of the BBC machines have given us comprehensive control over sound production, so some really exciting sounds can be produced.

There are four sound channels, which can be sounded singly, all together or as combinations of the four. At its simplest, we can produce a single note using the SOUND statement. Try these:

```
SOUND 0,-15,100,200
SOUND 1,-10,50,300
SOUND 2,-5,20,10
SOUND 3,-1,180,500
```

You can probably work out that the first figure tells the computer which sound channel you are using and that channel 0 is the "white noise" channel. The second figure determines the volume (-15 loudest to 0 for silence). The third and fourth control the pitch of the note and its length respectively.

If we alter the pitch, we get a rising or falling note. Try:

```
FOR I=0 TO 255:SOUND 1,-15,I,2:NEXT I
```

or

```
FOR I=255 TO 0 STEP-1:SOUND 1,-15,I,2:NEXT I
```

Now try sounding four channels at once:

```
SOUND0,-15,5,50:SOUND1,-15,50,50:SOUND2,-15,100,50:SOUND3,
-15,200,50
```

You should now be able to see how this siren program works. It sounds as if it is coming towards you since the volume is gradually increasing. See if you can amend the program to sound as if it is going away from you.

```
10 VOL=-1:DUR=3
20 FOR NEARER=1TO5
30 FOR PITCH=15 TO 0STEP-15
40 PROCNOISE(PITCH,VOL)
50 NEXT PITCH
60 VOL=VOL-3
70 NEXT NEARER
80 END
90 DEFPROCNOISE(PITCH,VOL)
100 SOUND1,0,0,0:SOUND2,0,0,0:SOUND3,0,0,0
110 SOUND1,VOL,PITCH+85,DUR*3:SOUND2,VOL,PITCH+133,DUR*3:
    SOUND3, VOL,PITCH+181,DUR*3
120 ENDPROC
```

(Hint: set VOL to -15 at line 10 and increase VOL by 3 in line 60.)

But, there are more complex ways of controlling the sound using the ENVELOPE statement. This is used before a SOUND statement to control the "shape" of the sound; for example, the change in volume. The ENVELOPE statement is followed by no less than 14 numbers each controlling a different aspect of the sound! For those readers who wish to know more about ENVELOPE, the BBC manual defines it on page 244. For those who are content to "meddle" with sound, here is a program that will define 8 different envelopes, each of which can be played on any or all of the four channels. You could try altering the numbers in the ENVELOPE statements for different effects.

```
5 REPEAT
6 MODE7
10 PROCSHAPE
30 PROCNOISE
40 UNTIL FALSE
1000 DEFPROCSHAPE
1010 ENVELOPE1,2,1,-1,0,2,2,0,4,0,0,-2,126,126
1020 ENVELOPE2,1,127,0,0,30,0,0,126,-1,0,-1,126,90
1030 ENVELOPE3,1,8,0,0,7,0,0,32,-1,0,-2,126,100
1040 ENVELOPE4,1,18,-4,5,7,5,5,30,-2,0,-2,126,100
1050 ENVELOPE5,5,16,0,0,7,0,0,32,-10,-2,-2,1267,100
1060 ENVELOPE6,4,63,65,29,49,56,208,30,0,-1,-10,126,126
1070 ENVELOPE7,5,28,20,0,2,1,0,0,-6,0,-1,126,100
1080 ENVELOPE8,5,-1,1,0,1,1,0,40,-3,-2,0,126,100
1090 INPUT" WHICH ENVELOPE NUMBER (1-8) "E
1100 ENDPROC
```

```
2000 DEFPROCCHANNEL
2010 INPUT " WHICH CHANNEL (0-3)";C
2020 ENDPROC
3000 DEFPROCNOISE
3010 *FX15,0
3020 FOR C=0 TO 3
3025 PRINT
3030 PRINT"CHANNEL ";C;" SOUNDS LIKE THIS:":SOUNDC,E,50*C,200
3035 PROCKEY
3040 *FX15,0
3045 NEXT C
3050 PRINT "ALL THREE TOGETHER SOUND LIKE THIS :"
3055 *FX15,0
3060 FOR C=0 TO 3
3070 SOUNDC,E,50*C,200
3080 NEXT C
3090 PROCKEY
3100 *FX15,0
3110 ENDPROC
4000 DEFPROCKEY
4005 FORI=1TO4000:NEXT
4010 PRINT'CHR$130"PRESS A KEY"
4020 A$=GET$
4030 ENDPROC
```

Finally, as an example of creative programming using sound, we are indebted to the BBC's 'Computer Programme' for this short program:

```
10 REM SEA SOUNDS
20 ENVELOPE1,5,0,0,0,5,2,3,10,-3,0,-1,126,70
30 ENVELOPE2,3,0,-2,-1,2,2,30,50,-1,-12,-2,126,126
40 REPEAT
50 surf=RND(2)
60 SOUND0,1,surf+3,20
70 gap=RND(20)+40
80 SOUND&1000,0,0,gap
90 IF RND(2)=2 THEN PROCtweet
100 UNTIL FALSE
110 DEF PROCtweet
120 flock=RND(5)
130 FORgulls=1TOflock
140 SOUND1,2,220,20
150 IFflock>1 THEN SOUND2,2,200,25
160 IFflock>2 THEN SOUND3,2,240,10
170 NEXT
180 ENDPROC
```

BBC Keyboard

This program uses some of the sound and graphics capabilities of the BBC computer to simulate a keyboard instrument.

The program begins by setting up two arrays to store the data for the keypresses and the notes to be played; the data is then read into the arrays.

After the screen is cleared by a MODE 7, the cursor is turned off and an ENVELOPE of sound defined (Piano). The keyboard is then drawn on the screen using Teletext graphics.

Then the main REPEAT loop of the program begins. After clearing any old keypresses (line 170) the program looks for a key being pressed. It then clears any notes still sounding (line 190) and finally checks through the array N$(I) to see whether any keypress made is a valid character. If it is then the corresponding value of the pitch is used into the SOUND channel (line 220). The actual sound is determined by the previous ENVELOPE statement.

```
10 DIM N$(22),T(22)
20 FORI=TO22:READ N$(I):NEXT
30 DATAA,S,D,F,G,H,J,K,L,";",":","]"
40 DATAQ,W,E,T,Y,I,O,P,"[","-"
50 FORI=1TO22:READ T(I):NEXT
60 DATA81,89,97,101,109,117,121,129,137,145,149,157
70 DATA85,93,105,113,125,133,141,153,161,173
80 MODE7
90 VDU23;8202;0;0;0;:PRINT
100 ENVELOPE1,1,1,-1,0,1,1,-1,127,-2,-1,-15,126,50
110 PRINT' ' " Q W E  T Y  I O P  [ -"
120 A$=CHR$156+CHR$151+CHR$157:B$=CHR$148+"5 "
130 FORI=1TO4:VDU148,255: PRINTA$A$A$B$A$A$B$A$A$A$B$A$
    CHR$156: NEXT
140 FORI=1TO5:VDU148,157,151:PRINTSTRING$(12,CHR$234+
    CHR$255+CHR$255):NEXT
150 PRINT"   A S D F G H J K L ; : ]"
160 REPEAT
170 *FX21,0
180 G$=GET$
190 *FX21,5
200 FORI=1TO22
210 IF G$<>N$(I) THEN230
220 SOUND 1,1T(I),10
230 NEXTI
240 UNTIL FALSE
```

The major amendments which can be made are of course in the shape of the sound as given by the ENVELOPE statement in line 100. Look up the parameters of the ENVELOPE statement, and change them then try out the program. Also try

changing the colour of the keyboard (148 in lines 120,130 and 140)

Electron users could omit the graphics or redesign them in MODE6.

Sound on the VIC-20

There are four sound channels on the VIC 20, which can be sounded singly, all together or in combinations. At its simplest, we can produce a single note using POKES. Try these:

```
POKE 36878,10
```

This sets the volume for our notes (0 is off 15 is loudest).

```
POKE 36874,150
```

This sound from the BASS channel can be cancelled by pressing RUN/STOP and RESTORE together!!

```
POKE 36875,150
```

This note comes from the ALTO channel.

```
POKE 36876,150
```

This note comes from SOPRANO channel.

```
POKE 36877,150
```

This note comes from the NOISE channel.

To sum up:

```
POKE 36874 for BASS sounds
POKE 36875 for ALTO sounds
POKE 36876 for SOPRANO sounds
POKE 36877 for NOISE
POKE 36878 for VOLUME
```

The value that we POKE into the sound channels alters the pitch, so we get a rising or falling note. Try:

```
FOR I=128 TO 254:POKE 36784,I:NEXT
```

or

```
FOR I=254 TO 128 STEP-1:POKE 36784,I:NEXT I
```

Now try sounding all four channels at once:

```
POKE36878,10:POKE36874,150:POKE36875,150:POKE36876,150:
POKE36877,150
```

You should now be able to see how this siren program works. It sounds as if it is coming towards you since the volume is gradually increasing. See if you can amend the program to sound as if it is going away from you.

```
10 FOR V=1 TO 15 STEP3
20 GOSUB1000
30 NEXT
40 END
1000 POKE 36878,V
1010 POKE 36875,235
1020 FORT=1TO600:NEXT
1030 POKE 36875,232
1040 FORT=1TO600:NEXT
1050 POKE 36875,0
1060 RETURN
```

(HINT: another call to the subroutine with the STEP command decreasing)

VIC KEYBOARD

This program uses some of the sound and graphics capabilities of the VIC to simulate a keyboard instrument.

The program begins by setting up two arrays to store the data for the keypresses and the notes to be played. The volume control is set, the data read and a keyboard drawn on the screen.

Then the main part of the program looks for a keypress. It then checks through the array N$(I) to see whether it is a valid character. If it is, then the corresponding value is POKED into the sound channel. This sound continues for a short time and is then turned off. A GO TO repeats the main loop.

N.B. As in all VIC programs in this book, the graphics characters have not been drawn, but the actual keys to obtain them have been used. For example, if you read [CLR] this means the button marked CLR. Now to get the heart-shaped character into the program, you will also have to use the SHIFT key at the same time, so the program shows [SHIFT+CLR].

```
10 DIM T(18),N$(18)
20 PRINT"[SHIFT+CLR]"
30 POKE 36878,10
40 FORK=1TO18:READ T(K):NEXT K
50 FORK=1TO18:READ N$(K):NEXT K
60 GOSUB1000
```

```
70 GET A$:IF A$="" THEN 70
80 FORI=1TO18
90 IF A$<>N$(I) THEN120
100 POKE 36875,T(I):FOR T=1TO50:NEXT T
110 POKE 36875,0
120 NEXT I
130 GOTO 70
140 DATA 135,147, 159,163,175,183,191,195,201,207,209
150 DATA 143,151,167,179,187,199,203
160 DATA Q,W,E,R,T,Y,U,I,O,P,@
170 DATA 2,3,5,6,7,9,0
1000 FORI=TO10:L$=L$+"[SPACE][SHIFT+H]":NEXT
1010 FORI=TO10:PRINTL$:NEXT
1020 FORI=TO22:B$=B$+"[COMMODORE KEY+T]:NEXT
1030 PRINTB$
1040 FORI=1TO4:C$=C$+"[CTRL+9][SPACE][CRSR DWN][CRSR RIGHT]":
     NEXT
1050 D$="[HOME][CRSR DOWN]"
1060 PRINTD$;TAB(1);C$;
1070 PRINTD$;TAB(3);C$;
1080 PRINTD$;TAB(7);C$;
1090 PRINTD$;TAB(9);C$;
1100 PRINTD$;TAB(11);C$;
1110 PRINTD$;TAB(15);C$;
1120 PRINTD$;TAB(17);C$;
1130 PRINT:PRINT"[3CRSR DOWN]Q[CRSR RIGHT]W[CRSR RIGHT]E[CRSR
     RIGHT]R[CRSR RIGHT]T[CRSR RIGHT]Y[CRSR RIGHT]U[CRSR
     RIGHT]I[CRSR RIGHT]O[CRSR RIGHT]P[CRSR RIGHT]@"
1140 PRINT"[HOME][SPACE]2[SPACE]3[3 SPACES]5[SPACE]6[SPACE]7
     [3SPACES]9[SPACE]0"
1150 RETURN
```

Try changing the volume in line 30, and the sound channel in lines 100+110. You could add a channel or even have a NOISE keyboard! Also, the duration of the note in line 100.

CHAPTER 6

Sorting and Filing Information

The computer, as we have seen, can store numbers, letters or words at identifiable locations or addresses. This happens all the time in programs, even with simple statements such as LET A = 4 or LET N$ = "JONES". We can call up the contents of the address A or N$ by entering PRINT A or PRINT N$. Of course, the hard part is all done for us by the computer!

We could store whole lists of numbers or words in variables like A or N$, but such lists would have little advantage over a written list. Fortunately we can use the computer to index and sort the numbers, letters or words, in various ways.

We are going to use a sorting routine developed by Donald Shell called Shellsort. But in order that we can appreciate and understand the beauty of Shell's idea, let us start with two earlier ideas: Bubblesort and Insertion sort.

Bubblesort

Suppose we want to sort these four numbers into ascending order:

9, 4, 7, 2

Bubblesort's first step is to compare the numbers in the first and second places. If they are already in ascending order, nothing more needs to be done, if not they are interchanged so:

9,4,7,2 becomes 4,9,7,2

The second step is to repeat the process with the number in the second and third place so that:

4,9,7,2 becomes 4,7,9,2

The third step is to compare the numbers in places three and four so that:

4,7,9,2 becomes 4,7,2,9

The end result of these three steps is to move the largest number to the last position.

The whole process is now repeated, except that we no longer need to the

comparison between the third and fourth position because the largest number is already in the fourth position.

4,7,2,9 remains as 4,7,2,9

but on the next step

4,7,2,9 becomes 4,2,7,9

The two largest numbers are now in the correct positions so that when the process is restarted the last two comparisons need not be made. The first step of the new process changes:

4,2,7,9 to 2,4,7,9

and the process is now complete.

This Bubblesort method is represented in BASIC by storing the numbers in an array. We can conveniently use a one-dimensional array, A, of length N, which will store our N numbers. In some versions of BASIC you must always declare the size of the array by a line such as 10 DIM A(20), which will store 20 numbers. In other versions, you only need to do this if DIMension exceeds 10. To sort the array into ascending order, we begin by comparing A(I), the i'th number in the list, with a(I+1). If A(I) is greater than A(I+1) then they are interchanged. We need to ensure that these comparisons are kept in order and completed. This is accomplished by two FOR/NEXT loops. In line 100, J counts the number of comparisons that are necessary and N-1 indicates that, each time, one fewer comparison is necessary.

The Bubblesort sub-routine is then:

```
100 FOR J = 1 TO N-1
110 FOR I = 1 TO N-J
120 IF A(I)=A(I+1) THEN 160
130 LET T = A(I)
140 LET A(I) = A(I+1)
150 LET A(I+1) = T
160 NEXT I
170 NEXT J
```

Don't forget that you may need to DIM the array if you are sorting.

Insertion sort

Let us suppose that we have a list of numbers A(1), A(2) ..... A(N) to be sorted into ascending order. We start with a list containing only one entry, A(1). The next term, A(2), is compared with A(1) and they are exchanged, if necessary, with A(1) to determine where it is to be inserted.

The routine for Insertion sort is:

```
100 FOR I=1 TO N-1
110 LET J=I
120 LET T=A(I+1)
130 IF T>=A(J) THEN 170
140 LET A(J+1)=A(J)
150 LET J=J-1
160 IF J>=1 THEN 130
170 LET A(J+1)=T
180 NEXT I
```

Shell sort

Donald Shell's idea was to move the smallest numbers to the left and the largest numbers to the right before using the insertion algorithm.

Suppose we have to sort the following set of numbers:

8, 2, 7, 4, 5, 3

The set is first divided into two groups: 8,2,7 and 4,5,3. The first number in the first group is compared with the first number in the second group. If it is bigger, it will be exchanged. Similarly the second and third numbers of each group are compared and exchanged if necessary, so that:

8, 2, 7, 4, 5, 3 would become 4, 2, 3, 8, 5, 7

This revised set of numbers is now divided into three groups of two numbers. Each number is compared and exchanged if necessary, so that:

4, 2, 3, 8, 5, 7 becomes 3, 2, 4, 7, 5, 8

The last step is to compare each number with its neighbour and to exchange them if necessary.

3, 2, 4, 7, 5, 8 becomes 2, 3, 4, 5, 7, 8

Each of these partial sorts has used the Insertion sort method.

The Shellsort routine for sorting N numbers is:

```
100 LET S=N
110 LET S=INT(S/2)
120 IF S>=1 THEN 130
130 FOR K=1 TO S
140 FOR I=K TO N-S STEP S
150 LET J=1
```

```
160 LET T=A(I+S)
170 IF T>=A(J) THEN 210
180 LET A(J+S) = A(J)
190 LET J=J-S
200 IF J>=1 THEN 170
210 LET A(J+S)=T
220 NEXT I
230 NEXT K
240 GO TO 110
```

Lines 100 to 130 are new to you. This Shellsort routine sets K to an integer from 1 to S. The S values are determined by successively dividing N by 2 (where N is the size of the set of numbers). Thus:

S=INT(N/2) and S=INT(N/4), S=INT(N/6) and so on until S=1

The Shellsort routine works much more quickly than either the Bubble or Insertion sort.

Handling files on BBC and VIC

Do not attempt this part of the book without having read the section in Chapter 2 that deals with DIMensioning arrays. There, a program for storing names and addresses is discussed - You will need to have understood that program before you can attempt this section.

Listings for an amended version of the address program appear below. The new versions of the program will allow you to store all the data on a cassette tape. If you made a copy of the original (from Chapter 2) then load it into your machine, and edit it so that it is the same as the ones below. Otherwise type in the whole program and SAVE a copy BEFORE you RUN it!!

Firstly, it is important to realise that when information is stored on a tape, it is in a linear format i.e. with one piece of data following the next. So to store the data from the address program, each name, address, telephone number and birthday has to be written onto the surface of the tape following each other. One way of imagining the access procedure is to think of the file as being in a cabinet. First you have to OPEN the file, then LOAD the file by taking it out of the cabinet. You can READ the file from one end to the other and stop when you have found the information you are looking for.

The BEEBFILER and VICFILER programs are menu-driven. This means that at the beginning you will be presented with various options and you make a choice. The options are:

SEARCH FILE. You cannot do this until you have fetched the file.

CREATE FILE. Do this when you first start

FETCH FILE. This gets the file you previously SAVEd on a data tape.

SAVE FILE. Do this when you have CREATEd your file.

END. If you mistakenly press 5 when you have not finished, try typing GOTO 41.

ALTER FILE. If someone on your file moves house, use this option to change the information.

Lines 10-20 set up arays for 20 people's names and data. The arrays are then filled with *'s to denote an empty entry. Flag is dropped.
Lines 41-45 form the menu.
Lines 46-50 wait for a number and then transfer control to the correct SUBroutine.
Line 60 returns to the menu.
Lines 75-550 form the search routine
Line 80 gets the search data
Line 85 checks for end of search data.
Line 90 starts the loop around the data
Line 150 sends control back to beginning of search routine.
Line 505 starts printout of data found.
Line 550 sends control back to search for more.
Lines 1000-1050 form the create file routine.
Line 1015 if END is inputted for name then the program returns to the menu.
Line 1044 checks for permissible maximum or flag raised.
Lines 2000-2050 form the fetch file routine.
Line 2000 opens the file for a READ.
Line 2020 starts the loop to INPUT the data from the tape file.
Line 2040 closes the file again.
Lines 3000-3050 form the save file routine.
Line 3000 opens the file for a WRITE.
Line 3020 starts the loop to PRINT the data onto the tape file.
Line 4000 ends the program.
Lines 5000-5060 form the alter file routine.
Line 5010 asks for the number of the entry to be changed.
Line 5020 raises the Flag, sets the entry number and calls the print data routine.
Line 5030 checks for any changes.
Line 5040 allows no changes and lowers the flag.
Line 5050 calls the input routine to amend data and lowers the Flag.

One of the constraints of writing a file handling program is lack of memory. Since most VIC users will have only 3K for both program and DATA, the listing contains no error trapping routines or fancy displays. This is something you could put in!

Another possible pitfall is the fact that the program has no data entry validation routine. This means that once you have pressed RETURN when creating a file, you are not given the chance to correct your mistake and the error goes into the array. The only way of amending the data is to use the ALTER FILE option 6 when you have reached the last person in your file (by typing END for the name input).

The program was written for a file of only 20 people. The amount or room you have depends on your type of computer, but try amending the program to increase the number of people on file.

A final warning - do *NOT* store data on the same cassette as the program! Store it on a separate cassette labelled ADDRESS DATA.

BEEBFILER

```
 10 DIM NAME$(20),TEL$(20),BIRTHDAY$(20)
 20  FORI=1TO20:NAME$(I)="*":ADDRESS$(I)="*":TEL$(I)="*":
   BIRTHDAYS$(I)="*":NEXT:FL=0
 41 CLS:PRINT"1. SEARCH FILE."
 42 PRINT"2. CREATE FILE."
 43 PRINT"3. FETCH FILE."
 44 PRINT"4. SAVE FILE.
 45 PRINT"5. END."
 46 PRINT"6. ALTER FILE."
 49 A=GET
 50 ON A-48 GOSUB 75,1000,2000,3000,4000,5000
 60 GOTO41
 75 CLS
 80 INPUT"SEARCH FOR ";SEARCH$
 85 IF SEARCH$="END"THEN RETURN
 90 FORI=1TO20
100 IF SEARCH$=NAME$(I) THEN 500
110 IF SEARCH$=ADDRESS$(I) THEN 500
120 IF SEARCH$=TEL$(I) THEN 500
130 IF SEARCH$=BIRTHDAY$(I) THEN 500
140 NEXTI
145 PRINT
150 PRINT"NO MORE FOUND":GOTO 80
500 PRINT"ENTRY ";I
505 PRINTNAME$(I)
510 PRINT"ADDRESS ";ADDRESS$(I)
520 PRINT"TELEPHONE ";TEL$(I)
530 PRINT"BIRTHDAY ";BIRTHDAY$(I)
540 PRINT
545 IF FL=1THENRETURN
550 GOTO140
999 REM *******CREATE FILE*******
1000 LET X=X+1:CLS:PRINT"ENTRY ";X
1010 INPUT"NAME ";NAME$(X)
1015 IF NAME$(X)="END"THEN1050
1020 INPUT"ADDRESS ";ADDRESS$(X)
1030 INPUT"TELEPHONE ";TEL$(X)
1040 INPUT"BIRTHDAY ";BIRTHDAY$(X)
1044 IF X=20 OR FL=1 THEN RETURN
1045 GOTO1000
```

```
1050 RETURN
1999 REM *******FETCH FILE*******
2000 F=OPENIN "ADDRESS":CLS:PRINT"LOADING ADDRESS FILE"
2010 FORI=1TO20
2020 INPUT#F,NAMES(I),ADDRESS$(I),TEL$(I),BIRTHDAY$(I)
2030 NEXTI
2040 CLOSE#0
2050 RETURN
2999 REM *******SAVE FILE*******
3000 F=OPENOUT "ADDRESS":CLS:PRINT"SAVING ADDRESS FILE"
3010 FORI=1TO20
3020 PRINT #F,NAME$(I),ADDRESS$(I),TEL$(I),BIRTHDAY$(I)
3030 NEXTI
3040 CLOSE#0
3050 RETURN
4000 END
5000 CLS
5010 INPUT"ENTRY NUMBER TO BE CHANGED ";I
5020 FL=1:X=I-1:GOSUB500
5030 INPUT"CHANGE (Y/N) ";Q$
5040 IF Q$="N" THEN FL=0:RETURN
5050 GOSUB1010:FL=0
5060 RETURN
```

VICFILER

```
 10 DIM N$*(20),A$(20),T$(20),B$(20)
 20 FORI=1TO20:N$(I)="*":A$(I)="*":T$(I)="*":B$(I)="*":NEXT:FL=0
 41 PRINT"[CLR]1. SEARCH FILE."
 42 PRINT"2. CREATE FILE."
 43 PRINT"3. FETCH FILE."
 44 PRINT"4. SAVE FILE."
 45 PRINT"5. END."
 46 PRINT"6. ALTER FILE."
 49 GET A:IF A=0 THEN 46
 50 ON A GOSUB 75,1000,2000,3000,4000,5000
 60 GOTO41
 75 PRINT"[CLR]"
 80 INPUT"SEARCH FOR ";S$
 85 IF S$="END"THEN RETURN
 90 FORI=1TO20
100 IF S$=N$(I) THEN 500
110 IF S$=A$(I) THEN 500
120 IF S$=T$(I) THEN 500
130 IF S$=B$(I) THEN 500
140 NEXTI
```

```
 145 PRINT
 150 PRINT"NO MORE FOUND":GOTO 80
 500 PRINT"ENTRY ";I
 505 PRINT"ADDRESS ";A$(I)
 520 PRINT"TELEPHONE ";T$(I)
 530 PRINT"BIRTHDAY ";B$(I)
 540 PRINT
 545 IF FL=1 THEN RETURN
 550 GOTO140
 999 REM CREATE FILE
1000 LET X=X+1:PRINT"[CLR]ENTRY ";X
1010 INPUT"NAME ";N$(X)
1015 IF N$(X)="END"THEN1050
1020 INPUT"ADDRESS ";A$(X)
1030 INPUT"TELEPHONE ";T$(X)
1040 INPUT"BIRTHDAY ";B$(X)
1044 IF X=20 OR FL=1 THEN RETURN
1045 GOTO1000
1050 RETURN
1999 REM FETCH FILE
2000 OPEN 1,1,0"ADDRESS":PRINT"[CLR]LOADING"
2010 FORI=1TO20
2020 INPUT #1,N$(I)
2021 INPUT #1,A$(I)
2022 INPUT #1,T$(I)
2023 INPUT #1,B$(I)
2030 NEXTI
2040 CLOSE 1
2050 RETURN
2999 REM SAVE FILE
3000 OPEN1,1,2,"ADDRESS":PRINT"[CLR]SAVING"
3010 FORI=1TO20
3020 PRINT #1,N$(I)
3021 PRINT #1,A$(I)
3022 PRINT #1,T$(I)
3023 PRINT #1,B$(I)
3030 NEXTI
3040 CLOSE 1
3050 RETURN
4000 END
5000 PRINT"[CLR]"
5010 INPUT"ENTRY NUMBER TO BE CHANGED ";I
5020 FL=1:X=I-1:GOSUB500
5030 INPUT"CHANGE (Y/N) ";Q$
5040 IF Q$="N"THEN FL=0:RETURN
5050 GOSUB1010:FL=0
5060 RETURN
```

File handling on the Spectrum

The least satisfactory feature of the Spectrum is the difficulty in saving programs that you wish to incorporate, later, into other programs. If you are using a tape recorder for file handling we suggest that you save the whole program and MERGE it with the original program when you want to use it again.

One of the best books on Spectrum file handling with a tape recorder is:-

David Lawerence. 'The Working Spectrum Volume 1' Sunshine Books ISBN 0946408 009

A more satisfactory method of file handling is to use a Microdrive. This program uses the Microdrive to keep a file of information about your friends:

```
 10 DIM N$(20) : DIM A$(20) : DIM T$(20) : DIM B$(20)
 20 FOR i = 1 TO 20
 21        LET N$(i) = "*"
 22        LET A$(i) = "*"
 23        LET T$(i) = "*"
 24        LET B$(i) = "*"
 25 NEXT i
 30 LET FL=0
 41 CLS:PRINT "1 ... Search File."
 42 PRINT "2 ... Create File."
 43 PRINT "3 ... Fetch File."
 44 PRINT "4 ... Save File."
 45 PRINT "5 ... End."
 46 PRINT "6 ... Alter File."
 49 INPUT "Put in your choice ";a
 50 IF a=1 THEN GOSUB 75
 51 IF a=2 THEN GOSUB 1000
 52 IF a=3 THEN GOSUB 2000
 53 IF a=4 THEN GOSUB 3000
 54 IF a=5 THEN GOSUB 4000
 55 IF a=6 THEN GOSUB 5000
 60 GO TO 41
 75 CLS
 80 INPUT "Search for "; S$
 85 IF s$= "end" THEN RETURN
 90 FOR i= 1 TO 20
100      IF s$ = N$(i) THEN GO TO 500
110      IF s$ = A$(i) THEN GO TO 500
120      IF s$ = T$(i) THEN GO TO 500
130      IF s$ = B$(i) THEN GO TO 500
140 NEXT i
145 PRINT
150 PRINT "No more found ": GO TO 80
500 PRINT "Entry "; i
```

```
 505 PRINT N$(i)
 510 PRINT "Address "; A$(i)
 520 PRINT "Telephone "; T$(i)
 530 PRINT "Birthday "; B$(i)
 540 PRINT
 545 IF FL=1 THEN RETURN
 550 GO TO 140
 999 REM *** CREATE FILE ***
1000 LET x=x+I: CLS : PRINT "Entry "; x
1010 INPUT "Name "; N$(x)
1015 IF N$(x)= "end" THEN GO TO 1050
1020 INPUT "Address "; A$(x)
1030 INPUT "Telephone "; B$(x)
1044 IF x = 20 OR FL = 1 THEN RETURN
1045 GO TO 1000
1050 RETURN
1999 REM *** FETCH FILE ***
2000 OPEN #4; "m";1; "filing "
2001 FOR i = 1 TO 20
2020         INPUT #4; LINE N$(i); LINE A$(i); LINE T$(i); LINE B$(i)
2030 NEXT i
2040 CLOSE #4
2050 RETURN
2999 REM *** SAVE FILE ***
3000 OPEN #4; "m";1; "filing"
3001 FOR i = 1 TO 20
3020 PRINT #4; N$(i)'A$(i)'T$(i)'B$(i)
3030 NEXT i
3040 CLOSE #4
3050 RETURN
4000 END
5000 CLS
5010 INPUT "Entry number to be changed'
5020 LET FL=1: LET x=i-1 : GOSUB 500
5030 INPUT "Change (Y/N) "; q$
5040 IF q$ = "N" THEN FL=0 : RETURN
5050 GOSUB 1010 : LET FL=0
5060 RETURN
```

CHAPTER 7

Machine Code

A simple way of looking at a computer is to imagine that it is just a collection of "boxes" or memory locations, each having its own identifying number (up to 65535). The processor inside (Z80 or 6502) organises the arrangement of what is to be stored where. So, when we decide on a variable name, N$ for example, the processor will assign a group of "boxes" in which to store the contents of variable N$. If N$ contains MICRO then the letters will be converted to hexadecimal codes 4D 49 43 52 4F and stored in a group of five memory locations, each having its own location number, or address.

	M	I	C	R	O
	&4D	&49	&43	&52	&4F
LOC	1E38	1E39	1E3A	1E3B	1E3C

Hexadecimal, you'll remember, is a shorthand way of writing down binary numbers. Hexadecimal counts from 1 to 9 as in decimal, but then uses A,B,C,D,E,F to represent numbers from 10 to 15.

Each time a BASIC program uses the variable N$, the processor has to go to the correct memory locations and take a copy of what is contained in them for use in the program.

BASIC is an "interpreted" language which means that whatever we type in is examined by the processor, checked for correct syntax and then converted into a simpler form for the processor to understand. BASIC is a "high level" language because it is one step away from the real processing.

Machine code, or assembly language programming, involves communicating with the machine's processor directly.

Let's look at a new way of programming the computer to add two numbers together in BASIC. Be sure to type in the program that is specific to your machine:

```
 5 REM BBC
10 INPUT A
20 ?&1D00=A
30 INPUT A
40 A=A+?&1D00
50 PRINTA
```

```
  5 REM COMMODORE VIC-20
 10 INPUT A
 20 POKE 2100,A
 30 INPUT A
 40 A=A+PEEK(2100)
 50 PRINTA

  5 REM SPECTRUM
 10 INPUT A
 20 POKE 23800,A
 30 INPUT A
 40 A=A+PEEK 23800
 50 PRINTA
```

Line 10 asks for a number
Line 20 stores that number in a "box"
Line 30 asks for a second number
Line 40 adds that number to the stored one
Line 50 prints the result

Although the BBC machine has a slightly different way of saying "store at box number", POKE 2100,A and ?2100=A mean the same thing - store A at box 2100. Similarly A+PEEK(2100) and A+?2100 both mean "add whatever is in box 2100 to A".

RUN the program several times and find out

What happens if you enter decimal numbers?
What happens if you enter negative numbers?
What happens if you enter letters?
What happens if you enter very large numbers?

You should find that the program will only give the correct answer if you input positive whole numbers less than 255 (&FF). One of the reasons for this is that the highest number the computer can store in one box is 255. Negative numbers are stored as "two's complement" and decimal numbers need more than one box to be stored. You could store letters if you changed the program to store the ASCII code instead.

Programming in machine code is almost like trying to write a BASIC program with only three variables and with no error checking when it is run. Make a mistake in BASIC and up comes a friendly message like SYNTAX ERROR. But not in machine code! What happens is that the machine crashes and the processor simply chases around its memory "boxes" getting lost and won't give control back to you!

Right at the beginning of this book, you read how the computer stores a BASIC program as a series of bytes; machine code programs are stored in the same way. It would be very tedious (and almost impossible) if each program had to be written in hexadecimal code, so each machine will have a BASIC program called an

Assembler written for it, which makes things easier. If you have a BBC machine, then the Assembler is already in the machine when you switch on. Other machines need to have the Assembler loaded from tape or disc.

Three important memory locations are the ACCUMULATOR, the X REGISTER and the Y REGISTER. These are the variables available to the machine code programmer.

Let's look at how the previous BASIC program to add two numbers together might be written in machine code.

LDA #& 0A	LoaD Accumulator with 10
STA &1D00	STore Accumulator at box &1D00
LDA #& 05	LoaD Accumulator with 5
ADC &1D00	ADd to the accumulator (with Carry) the contents of box &1D00
STA &7C00	STore Accumulator in box &7C00 (on the screen in BBC machines)

An alternative way of producing machine code is to buy a machine - specific program called a compiler, which has been designed to convert your BASIC program into Machine Code or vice versa. You will see many different compilers designed to do this on your computer. Some make extravagant claims, so always test the compiler to see that it really will do what you want it to do. Such a compiler can be very useful in developing machine code games programs, which are much faster than their BASIC equivalents.

Of course, with a compiler you are not actually writing in machine code; you write your program in BASIC, and the compiler does the rest!

If you are interested in machine code programming then the following books are recommended as a good introduction:

Discovering BBC Micro Machine Code by A.P. Stephenson, published by Granada (ISBN 0 246 121602).

Introducing Spectrum Machine Code by I.R. Sinclair, published by Granada (ISBN 0 246 12082 7)

VIC 20 Exposed (chapters on machine code) by John Vader Reyden, published by Melbourne House (ISBN 0 86161 132 2).

CHAPTER 8

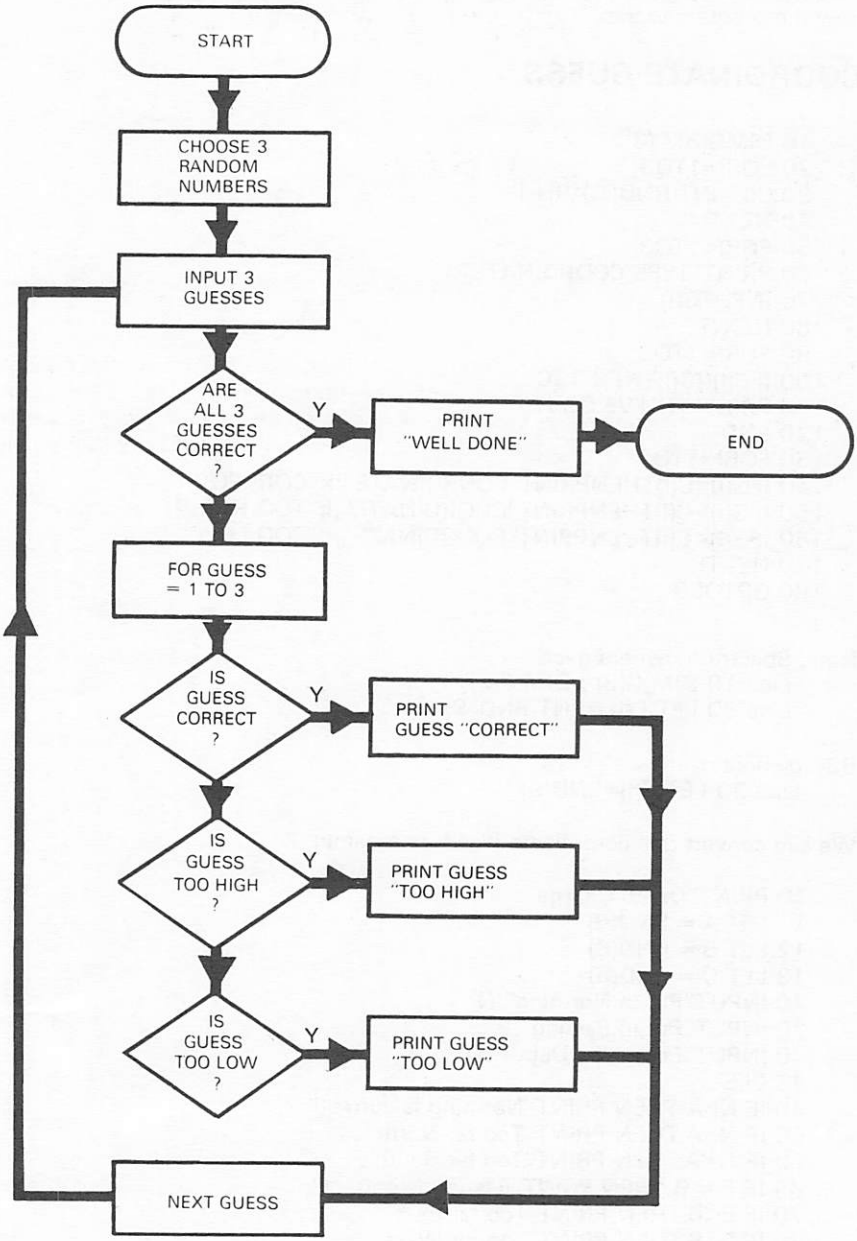
Fun and Games

One person's game is sombody else's learning program - you can learn quite a bit about computers by playing Space Invaders, but a lot more by writing your own games programs.

To illustrate this, we begin by describing a simple guessing game. At its simplest level, you just guess three numbers. But, these three numbers could be used to locate a submarine under the surface of the sea, so you then have to think in geometrical terms. It is only a short step from there to a program from which you'll learn some geometry. Hang on - you'll see!

THE 3-D GUESSING GAME

Here is a flowchart for the game:


START
CHOOSE 3 RANDOM NUMBERS
INPUT 3 GUESSES
ARE ALL 3 GUESSES CORRECT ?
Y
PRINT "WELL DONE"
END
FOR GUESS = 1 TO 3
IS GUESS CORRECT ?
Y
PRINT GUESS "CORRECT"
IS GUESS TOO HIGH ?
Y
PRINT GUESS "TOO HIGH"
IS GUESS TOO LOW ?
Y
PRINT GUESS "TOO LOW"
NEXT GUESS

Compare this program with the flowchart above. Trace the paths through it to see how the program works.

COORDINATE GUESS

```
 10 DIMG(3),C(3)
 20 FORI=1TO3
 30 C(I)=INT(RND(TI)*9)+1
 40 NEXTI
 50 FORI=1TO3
 60 PRINT"TYPE COORDINATE";I
 70 INPUTG(I)
 80 NEXTI
 90 FORI=1TO3
100 IFC(I)()G(I)THEN 130
110 PRINT"YOU'VE GO IT!!"
120 END
130 FORI=1TO3
140 IFG(I)=C(I)THENPRINT"COORDINATE";I;"CORRECT"
150 IFG(I)>C(I)THENPRINT"COORDINATE";I;"TOO HIGH"
160 IFG(I)<C(I)THENPRINT"COORDINATE";I;"TOO LOW"
170 NEXTI
180 GOTO50
```

Note, Spectrum owners need
Line 10 DIM G(3) : DIM C(3)
Line 30 LET C(I) = INT RND*9)+1

BBC owners need
Line 30 LET C(I)=RND(9)

We can convert this co-ordinate idea into a game:

```
 10 PRINT"Depth Charge"
 11 LET A = RND(6)
 12 LET B = RND(6)
 13 LET C = RND(6)
 20 INPUT"Put in Northing";N
 30 INPUT"Put in Easting";E
 40 INPUT"Put in the Depth";D
 45 CLS
 49 IF N=A THEN PRINT"Northing is correct"
 50 IF N>A THEN PRINT"Too far North"
 60 IF N<A THEN PRINT"Too far South"
 69 IF E = B THEN PRINT"Easting is correct"
 70 IF E<B THEN PRINT"Too far East"
 80 IF E>B THEN PRINT"Too far West"
 89 IF D = C THEN PRINT"Depth is correct"
```

```
 90 IF D<C THEN PRINT"Too shallow"
100 IF D>C THEN PRINT"Too deep"
130 IF N = A AND E = B AND D =C THEN PRINT"SUNK"
140 GOTO 20
```

VIC owners need to change lines 11 to 13 to

```
11 LET A=INT(RND(1)*6)+1 etc
```

Spectrum owners need the same modification, but omitting the (1) after RND.

Now let's look at a more serious "game". Actually, it's a simple language called LOGO that you can make your computer understand, as a change from BASIC.

Simple LOGO

Most of you will have heard of or even used the "turtle" language LOGO, devised by Seymour Papert. The children's toy Bigtrak is another implementation of the idea of programming an object to trace out a path decided by the operator.

Now the job of programming the screen cursor to build up intricate patterns is a very difficult and time consuming one. LOGO was developed to make the task simpler. LOGO is itself a computer program which can be programmed! The instructions to the cursor are much simplified so that a complex shape could be built up and then stored, each part of the shape becoming a subroutine or procedure which could be called by the main program (as in structured programming). So the instructions to draw a square could be something like this:

```
REPEAT 4
FORWARD 10
LEFT TURN 90
```

This subroutine could then be given a label such as WINDOW and later called up in a program to draw a house.

The program we will design is much simpler and less sophisticated. Those readers who are really interested will be able to purchase a LOGO interpreter for their machine from a good software distributor.

Because of the differences in the graphic capabilities of each machine, the main program given below is written for the VIC and amendments for Spectrum and BBC follow it.

The structure for our simple LOGO is:

INITIALISE VARIABLES

↓

GET PLOT CHARACTER

↓

GET COMMAND

↓

IMPLEMENT COMMAND

Initialise variables

Y$ (VIC only) is the string of cursor downs that will be used to position the characters in the correct place on the screen.

X and Y contain the maximum allowable screen widths and depths possible.

V will be used to store the number of places the cursor has to move.

I is used as a general purpose counter.

R$ is a string of delete characters used for rubbing out unwanted text.

P$ will hold the character to be used for printing on the screen.

S$ is used as a dummy variable to store P$ temporarily, should any unplotting be required.

CM$ will hold the command for the next plot.

D$ holds the first character of the command string.

V holds the value of the number of the last character of the command string.

VIC LOGO

```
10 GOSUB100
20 GOSUB160
30 GOSUB210
40 GOSUB360
50 GOTO30
60 :
70 :
80 REM ************************
90 REM ***INITIALISE VARIABLES***
```

```
100 PRINT "[CLR]"
105 Y$="[22 CRSR DOWN]"
110 LET X=11:LET Y=12:LET V=0
120 FOR I=1 TO5:R$=R$+CHR$(20):NEXT
130 RETURN
140 REM *************************
150 REM ****GET PLOT CHARACTER****
160 PRINT"[HOME]";LEFT$(Y$,20);:INPUT"PLOT CHARACTER ";P$
170 LET S$=P$
180 RETURN
190 REM *************************
200 REM ********GET COMMAND*******
210 PRINT"[HOME]";LEFT$(Y$,21);"COMMAND    ";R$;:INPUT CM$
220 LET D$=LEFT$(CM$,1):LET V=VAL(RIGHT$(CM$,1))
230 RETURN
240 REM *************************
250 REM ******PLOT COMMAND********
260 FORI=1TOV
270 IF D$="U"THEN LET P$=" ":REM UNPLOT
280 IF D$="P"THEN LET P$=S$ :REM PLOT
290 IF D$="C"THEN GOSUB 160 :REM CHANGE CHARACTER
300 IF D$="N"THEN LET Y=Y-1 :REM NORTH
310 IF D$="S"THEN LET Y=Y+1 :REM SOUTH
320 IF D$="E"THEN LET X=X+1 :REM EAST
330 IF D$="W"THEN LET X=X-1 :REM WEST
340 IF D$="F"THEN END
350 IF Y<0   THEN LET Y=0
360 IF Y>19  THEN LET Y=19
370 IF X<0   THEN LET X=0
380 IF X>21  THEN LET X=21
390 PRINT"[HOME]";LEFT$(Y$,Y);TAB(X);P$
400 NEXTI
410 CM$=""
420 RETURN
```

BBC LOGO AMENDMENTS

```
100 MODE6
105 DO NOT TYPE THIS LINE
110 LET X=20:LET Y=12:LET V=0
120 FORI=1TO10:R$=R$+CHR$(8):NEXT
160 PRINTTAB(20,23);:INPUT"PLOT CHARACTER "P$
210 PRINTTAB(0,23)"COMMAND      ";R$;:INPUT CM$
360 IF Y>22 THEN LET Y=22
380 IF X>89 THEN LET X=39
390 PRINTTAB(X,Y);P$
```

```
100 CLS
105 DO NOT TYPE THIS LINE
110 LET X=16:LET Y=12:LET V=0
120 FORI=1TO5:R$=R$+CHR$(12):NEXT I
160 PRINT AT 20,0:INPUT"PLOT CHARACTER ";P$
210 PRINT AT 21,0 "COMMAND    ";R$;:INPUT C$
220 LET D$=C$(TO 1):LET V=VAL (C$(2 TO 2))
380 IF X>31 THEN LET X=31
390 PRINT AT X,Y;P$
```

GET PLOT Character

A feature of our LOGO is that the shape drawn on the screen can be any character that the computer can print. So, you could draw a shape with *'s or .'s etc. VIC owners will be at a particular advantage here because their keyboard has many predefined graphics characters.

Spectrum and BBC owners can both print text on the screen at a certain location with the PRINT AT and the PRINT TAB statements respectively. VIC owners will find that although their machine can PRINT TAB, this only refers to the horizontal direction. Thus the program has to define a string (Y$) which will position the text "COMMAND" 21 lines down the screen. So PRINT"[HOME]";LEFT$(Y$,21); is the same as PRINT TAB(0,21).

GET command

To simplify the program, the commands entered are just two characters long. So, N7 means North 7 places. Acceptable inputs are N,S,W, and E together with a number up to a maximum value of 9. Extra commands allow you to Unplot, Plot, Change plot character and Finish.

PLOT command

This subroutine is really a loop using the index variable I count from 1 to V, the number of plots the user has commanded.

The first character of the command is checked,

if U then spaces are plotted (Unplot).
if P then the last character (in S$) is plotted.
if C then the Get character subroutine is visited.
if N then 1 is deducted from the Y coordinate.
if S then 1 is added to the Y coordinate.
if E then 1 is added to the X coordinate.
if W then 1 is deducted from the X coordinate.
if F then the program Ends.

Lines 350 to 380 are to stop screen "wrap round" which would spoil the display.
Line 390 prints the character at the correct place on the screen and before returning from the subroutine, the command string is emptied.

When you have tried the program a few times and are familiar with the way it works, try drawing some recognisable shapes, making note of the instructions.

Amendments could include:

(1). Adding the available directions North West, North East etc., either by giving each a one letter code (e.g. Q for North West) or rewriting the program to check for the first two letters of the command string.
2. Adding the option of printing in colour.
3. Adding sound to the plotting.
4. Trying to write the plot subroutine using high resolution graphics.

SCHOOL ADVENTURE

This program will provide you with the framework for an adventure-type game, but with a difference! The aim of the game is to reach the tenth floor of the school with ten house points.

On the way to the top you will meet several teachers who ask you questions. Answer correctly and you will gain one housepoint; an incorrect answer means one deducted.

You can, however, give the answer UP to any teacher's question - this means you will go up a floor *but* you will lose five housepoints. When you reach room five on any floor you will automatically go up the stairs to the next floor.

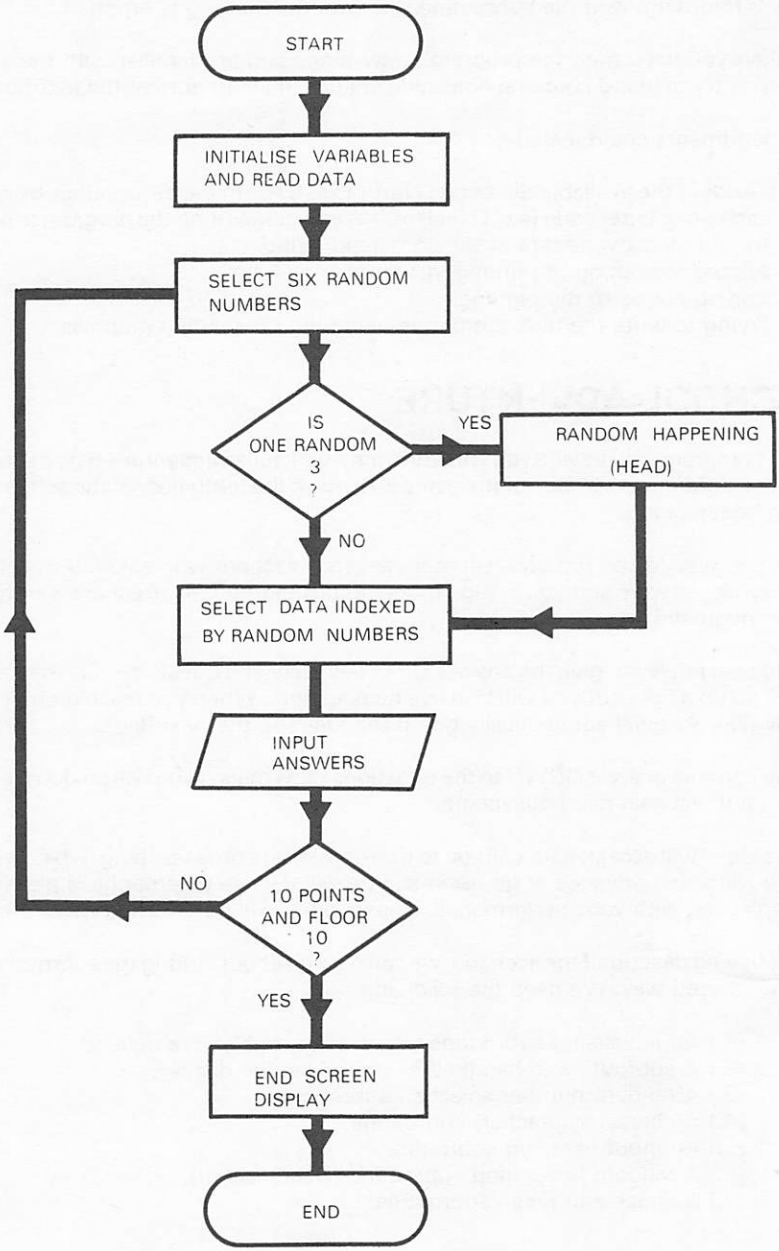
You can also answer DOWN to the questions - this means you will go down a floor *but* you will gain five housepoints.

The Head will occasionally call you to the room where he is teaching (which means you will either advance or go back) and he will tell you whether he is pleased or displeased with your performance. Housepoints will be awarded accordingly!

So, having described the scenario, we can now go about writing the subroutines in a structured way. We need the following:

(1.) An initialising subroutine where the variables are defined.
(2.) A subroutine to handle the "room" i.e. the display.
(3.) A random number selection subroutine.
(4.) A character (teacher) subroutine.
(5.) An input handling subroutine.
(6.) A random happening subroutine (Headteacher).
(7.) A check and finish subroutine.

Here is the flow chart of the structure:



```
  10 GOSUB1000: REM init
  20 GOSUB2000: REM room
  30 GOSUB3000: REM rand
  40 GOSUB4000: REM teacher
  50 GOSUB5000: REM answer
  60 IF HP>=10 AND F=10 THEN GOSUB 7000:END: REM finish
  70 GOTO20
  80 END
  90:
 100:
 990 REM SUBROUTINE init;Spectrum needs separate DIMs and single
     letter variables
1000 DIM X(6),TE$(5), OB$(5),QU$(5),AN$(5)
1010 REM CHANGE 5'S WHEN ADDING MORE DATA
1020 LET R=1:LET F=0:LET HP=0
1030 FORI=ITO5: REM CHANGE 5 WHEN ADDING MORE DATA
1040 READ TE$(I),OB$(I),QU$(I),AN$(I)
1050 NEXT
1060 RETURN
1990 REM SUBROUTINE room
2000 CLS:REM VIC PRINT"[CLR]" ALSO ADD GRAPHICS + SOUND HERE
2010 PRINT"YOU ARE ON FLOOR ":F
2020 PRINT"IN ROOM ";R
2030 PRINT"HOUSE POINTS ";HP
2040 RETURN
2990 REM SUBROUTINE rand
3000 FOR I=1 TO 4
3010 LET X(I)=RND(5): REM VIC LET X(I)=INT(RND(1)*5+1). Spectrum
     similar but omit (1)
3020 REM CHANGE 5 WHEN ADDING NEW DATA
3030 NEXT
3040 LET X(5)=RND(5): REM VIC LET X(I)=INT(RND(1)*5+1). Spectrum
     omit (1)
3050 LETX(6)=RND(2): REM VIC LET X(I)=INT(RND(1)*2+1). Spectrum
     omit (1)
3060 IF X(4)=3THEN PRINT:PRINT"The Head wants you in room ";X(5):
     GOSUB6000
3070 RETURN
3990 REM SUBROUTINE teacher
4000 PRINT:PRINT"Along comes ";TE$(X(1));","
4010 PRINT"with a ";OB$(X(2))
4020 PRINT
4030 PRINTQU$(X(3))
4040 PRINTTE$(X(1));" asks."
4050 RETURN
4990 REM SUBROUTINE answer
5000 PRINT
5010 PRINT"WHAT IS YOUR ANSWER ";:INPUT G$
5020 IF G$=AN$(X(3))THEN LET HP=HP+1:LET R=R+1
```

```
5030 IF G$<>AN$(X(3))THEN LET HP=HP-1
5040 IF G$="UP"THEN LET F=F+1:LET HP=HP-5
5050 IF G$="DOWN"THEN LET F=F-1:LET HP=HP+5
5060 IF R=6 THEN LET R=1:LET F=F+1
5070 IF F<0 THEN F=0
5080 IF F>10 THEN F=10
5090 IF R<1 THEN R=1
5100 RETURN
5990 REM SUBROUTINE head
6000 PRINT
6010 FORI=1TO2000:NEXT
6020 LET R=X(5)
6030 IF X(6)=1 THEN GOTO6060
6040 PRINT"I am not pleased, lose ";x(5);" housepoints"
6050 LET HP=HP-X(5):GOTO6080
6060 PRINT"I am very pleased, take ";x(5);" housepoints"
6070 LET HP=HP+X(5)
6080 FOR I=1TO2000:NEXT
6090 RETURN
6990 REM SUBROUTINE finish
7000 CLS:PRINT"WELL DONE":REM VIC PRINT"[CLR] WELL DONE"
7010 RETURN
8000 DATA "Mr.Rowling";"book."
8010 DATA "What is the formula for water?","H20"
8020 DATA "Mr.Dolphin","computer."
8030 DATA "What keyword goes with DATA?","READ"
8040 DATA "Mrs.Bretherton","pair of trainers."
8050 DATA "What is 1094 * 365?","399310"
8060 DATA "Mrs.Leadbetter","typewriter."
8070 DATA "Who is the father of computing?","BABBAGE"
8080 DATA "Mr.Crosby","box of crisps."
8100 DATA "When was the battle of Hastings?","1066"
8110 REM ADD MORE DATA HERE
```

FURTHER SUGGESTIONS

Personalise the program by substituting your own teacher's (or friend's) names in place of the ones used in the listing. You can also change the questions and the objects, but when making changes be sure that your DATA lines contain the question immediately followed by the correct answer. (You will also have to decide how you will deal with inputs such as BABBAGE, Babbage, Charles Babbage etc.)

Add more DATA after line 8100. The amount of extra DATA you can fit in will depend on the machine you are using and in the case of the VIC 20, whether you have memory expansions fitted. Adding more DATA, means that the number 5 in lines 1000,1030 and 3010 will have to be changed to match the total number of characters in the DATA statements. It is always wise to save a version of the new program with the extra DATA *before* you run it, in case you lose the program for some reason.

The program works by storing your message in a matrix which is defined by the DIM statement. This is as large as the length of your message. The computer then selects a random number and positions each letter in that random position in the matrix. Then it simply scans the grid, from left to right, printing out what it finds in each of the cells of the matrix.

Line 50 works out how long your message is.
Line 60 sets up the matrix.
Line 80 selects random numbers.
Line 90 checks to see if the selected cell in the matrix has already been filled.
Line 100 puts a letter in the selected cell.
Line 190-210 - prints out the contents of the matrix.

ANAGRAM

This program will produce an anagram of a string which you input. The program begins by asking you for the message. Then the computer will display a computed anagram using the letters and spaces which you inputted. Try something like "COMPUTERS ARE STUPID" and see the results!

```
 10 CLS:REM VIC PRINT "[CLR]"
 20 PRINT"TYPE YOUR MESSAGE"
 30 PRINT
 40 INPUT M$
 50 LET M=LEN(M$)
 60 DIM I$(M)
 70 FORI=1TOM
 80 LET P=RND(M):REM VIC LET P=INT(RND(1)*M)+1; Spectrum omit
    (1)
 90 IF I$(P)<>""THEN GOTO 80
100 I$(P)=MID$(M$,I,1):REM SPECTRUM I$(P)=M$(I TO I)
110 NEXT I
120 PRINT
130 PRINT"MY ANAGRAM OF ----"
140 PRINT
150 PRINTM$
160 PRINT
170 PRINT"IS -----"
180 PRINT
190 FORI=1 TOM
200 PRINTI$(I);
210 NEXT I
220 PRINT:PRINT
230 PRINT"--------"
240 PRINT
250 CLS:REM VIC PRINT "[CLR]"
260 GOTO20
```

Introduce a time element. You could set the internal clock of your computer to zero at the beginning of the program and keep checking it during the running of the program. Then, if it exceeds a value which you decide (say 3 minutes) the sound generator could buzz and the player is transported to a random room on a random floor.

We have included objects in the program, but as it stands, they have no significance. You could introduce another subroutine so that if the teacher was carrying say, the computer, a different test of the player's skill is made.

The final screen display simply says WELL DONE. Make this more attractive by using your computer's graphics and sound capabilities to provide a really effective reward!

SPEED READ

This program is a test of how accurately you can speed read (and type!).

A sentence is displayed for a short time on the screen you are asked to type in what you read. If you are wrong the computer will show you the correct sentence plus the sentence you typed in.

The program begins by asking you for the speed you wish to play - 0 is very fast, 10 is very slow.

```
 10 CLS:REM VIC PRINT"[CLR]"
 20 PRINT"HOW FAST (0-10)";
 30 INPUT T
 40 READ S$
 50 IF S$="END" THEN END
 60 CLS:REM VIC PRINT"[CLR]"
 70 PRINTS$
 80 FORI=1 TO 1000*T:NEXT
 90 CLS:REM VIC PRINT"[CLR]"
100 PRINT"WHAT DID I PRINT?"
110 PRINT
120 INPUT G$
130 IF G$=S$ THEN PRINT"GOOD":GOTO40
140 PRINT"I PRINTED -----"
150 PRINT
160 PRINT$
170 PRINTG$
180 PRINT
190 PRINT"----- IS WHAT YOU TYPED"
200 PRINT:PRINT"PRESS A KEY"
210 A$=GET$:REM VIC GETA$:IF A$=""THEN 210:REM SPECTRUM
    PAUSE 0
```

```
220 GOTO40
230 DATA "THE CAT SAT ON THE MAT"
240 DATA "COMPUTERS SAVE TIME"
250 DATA "READING FAST IS HARD"
1000 DATA "END"
```

The program works by displaying a sentence which it has READ from the DATA for a time computed from the INPUT. A check is then made that the displayed string (S$) is the same as the INPUTted string (G$) at line 130.

There are many improvements which could be made to this program! You could:

Add a lot more DATA to the program before line 1000.

Make the sentence displayed a random choice, then you would not know which was coming next. For this you would need to READ all the DATA first and then select each one at random.

Devise a scoring system. Use C=C+1 every time a correct input is made and W=W+1 for wrong inputs. The score could be displayed at the top or bottom of the screen.

Increase the length of time that the sentence is displayed if the person gets it wrong AND decrease the time if he is right.

Put in a timing system so that the person can see how long he took to type in the sentence.

Be sneaky and put in punctuation to catch people out!

CHAPTER 9

Your Turn Now!

Several years ago, many computer scientists started to attack the BASIC programming language, saying it was too "unstructured" and allowed programmers to write muddled programs.

The term "spaghetti programs" was coined to describe those programs which contained lots of IF's and GOTO's. Trying to follow the logic of one of these programs is rather like trying to trace a strand of spaghetti from one end to the other when it is all coiled up!!

As one part of the suggested alternatives, COMAL (COmmon Algorithmic Language) was devised by a man called Loefstedt. This language encourages programmers to take a more structured approach by using such keywords as:

REPEAT...UNTIL
PROC...ENDPROC,
CASE...WHEN...OTHERWISE...ENDCASE.

BBC machine users will recognise that BBC Basic has incorporated the first two of these useful "structured keywords" into its repertoire.

The way to attempt the following projects is to take the structured approach and follow the steps given:

(1.) Draw some cloud diagrams to show what will happen.

(2.) Divide what is in each of the clouds into small parts.

(3.) Call each part by a suitable name.

(4.) Write the listing for each part ON PAPER.

(5.) Write a "driver" program to call each part in the correct order, by using GOSUB or PROC, but again ON PAPER.
(6.) Now type the whole program into the computer and RUN it.

(7.) Now correct the errors!!!

But remember Murphy's Law of Computing -

"THERE IS ALWAYS ONE MORE BUG!!"

Projects

Project 1. Write a program to play the game where the user is asked to think of a number, double it, add 10, divide by 2 etc. Then the computer prints the number that was chosen.

Project 2. Write a program which will work out how many days it would take for you to earn £1 million, if you were paid 1p on the first day, 2p on the second, 4p on the third, 8p on the fourth etc.

Project 3. Write a program to help you practise Morse Code.

Project 4. Write a program to calculate currency conversions for continental holidays.

Project 5. Can you produce a light-hearted and amusing fortune-telling program?

Project 6. Devise a program to play a game where the computer selects a number and the user has to find it by asking whether it is higher or lower than a number of their choice. A program similar to this one can be found in chapter 3.

Project 7. There is a three digit positive number which:

if you add 7 to it, it divides exactly by 7
if you add 8 to it, it divides exactly by 8
if you add 9 to it, it divices exactly by 9
Write a program to find the number

Project 8. My four figure telephone number was recently changed by British Telecom. Three facts, however, make it easy to remember my old telephone number:

the new number is exactly four times the old number

the new number is the old number written back to front

neither number contains a zero

Can you write a program to find my old number and my new number?

Project 9. See if you can write a program to solve this puzzle:

Which ten digit perfect number contains the most zeroes?

Which ten digit perfect square number contains the most ones and so on ?

Project 10. Can you write a program to make the computer count to a million in as short a time as possible ?

Project 11. Five sailors and a monkey are shipwrecked on an island whose only source of nourishment is an abundance of coconuts.
On their first day they gather together as many coconuts as they can find and place them in a very large heap, intending that on the following day they will share them out fairly between the five sailors.
During the night, the first sailor awakes and (worried about the honesty of his fellow-sufferers) decides to take his share. He divides the heap into five but finds a remainder of 1 coconut which he gives to the monkey. He hides his 'share' and leaves the rest in a single pile.
Later that night, the second sailor also decides to take his share, unaware of the actions of the first. He too finds that in attempting to divide the pile into five that there is a remainder of 1 coconut. He too gives the odd one to the monkey, and then takes his 'share'.
In turn, each of the other three sailors do the same, with the same result, and without each other's knowledge .
The following morning, the five sailors (not wishing to show their mistrust of each other) agree to divide the heap (now much smaller of course) into five shares. Once again there is one remaining which is given to the monkey.

What is the smallest number of coconuts in the original pile to have made this account possible???

Suggested Answers.

Spectrum owners will need to use separate DIM statements and to change any two letter variables to one letter ones in each of the following programs.

PROJECT 1.

```
 10 PRINT"CHOOSE TWO NUMBERS EACH LESS THAN TEN":GOSUB1000
 20 PRINT"DOUBLE ONE OF THE NUMBERS":GOSUB1000
 30 PRINT"NOW ADD FOUR TO YOUR ANSWER":GOSUB1000
 40 PRINT"NOW MULTIPLY YOUR ANSWER BY FIVE":GOSUB1000
 50 PRINT"NOW ADD THE OTHER NUMBER YOU THOUGHT OF":
    GOSUB1000
 60 INPUT"WHAT DO YOU GET ";N
 70 LET N=N-20
 80 LET N$=STR$(N)
 90 LET N1$=LEFT$(N$,1)
100 LET N2$=RIGHT$(N$,1)
110 PRINT"YOUR TWO NUMBERS WERE ";N1$;
120 PRINT" AND ";N2$
130 END
1000 FOR I=1 TO 5000:NEXT:RETURN
```

PROJECT 2.

```
 10 LET P=0.01                          :REM START WITH 1 PENNY
 20 LET P=2*P                                  :REM DOUBLE IT
 30 LET DAY=DAY+1                              :REM NEXT DAY
 40 PRINT"DAY ";DAY,"MONEY ";P           :REM PRINT RESULTS
 50 IF P>=1000000 THEN END             :REM TARGET REACHED
 60 GOTO20                                     :REM DO IT AGAIN
```

PROJECT 3.

```
 10 GOSUB100:REM INITIALISE
 20 GOSUB500:REM MAIN PROGRAM
 30 GOTO20
 99 REM******INITIALISE*********
100 DIM L$(26) ,MC$(26),W$(20)
110 FORI=1TO26:READ L$(I),MC$(I):NEXT
120 FORI=1TO20:READ W$(I):NEXT
130 RETURN
140 DATA "A",".-"
150 DATA "B","-...
160 DATA "C","-.-.
170 DATA "D","-.."
180 DATA "E","."
190 DATA "F","..-."
200 DATA "G","--."
210 DATA "H","...."
220 DATA "I",".."
230 DATA "J",".---"
240 DATA "K","-.-"
250 DATA "L",".-.."
260 DATA "M","--"
270 DATA "N","-."
280 DATA "O","---"
290 DATA "P",".--."
300 DATA "Q","--.-"
310 DATA "R",".-."
320 DATA "S","..."
330 DATA "T","-"
340 DATA "U","..-"
350 DATA "V","...-"
360 DATA "W",".--"
370 DATA "X","-..-"
380 DATA "Y","-.--"
390 DATA "Z","--.."
400 DATA "APPLE","BAKER","CARAVAN","DRAMA"
410 DATA "EVERY","FLAKES","GRAZE","HOUSES"
420 DATA "INDIGO","JOYFUL","KITES","LORRIES"
430 DATA "MAXIMUM","NEITHER","OPAQUE","PEACE"
```

```
440 DATA "QUICK","RHYTHM","SNOOZE","TRIANGLE"
499 REM*****MAIN PROGRAM********
500 PRINT"TYPE A FOR MORSE ALPHABET"
510 PRINT
520 PRINT"TYPE B FOR MORSE PRACTICE"
530 PRINT
540 INPUT"WHICH ONE ";C$
550 IF C$="A"THEN GOTO 580
560 IF C$="B"THEN GOTO 800
570 GOTO 540
580 FORI=1TO26
590 PRINTL$(I);" IN MORSE IS ";
600 GOSUB1000:REM SOUND
690 NEXT I
700 RETURN
800 LET X=RND(20):REM VIC X=INT(RND(1)*20)+1:SPECTRUM
    LETX=INT(RND)*20)+1
810 LET X$=W$(X)
820 PRINT"LISTEN TO THIS WORD"
830 FORT=1TO LEN (X$)
840 LET D$=MID$(X$,T,1):REM SPECTRUM LET D$=X$(T TO T)
850 FORI=1TO 26
860 IF D$=L$(I)THEN 880:REM-Spectrum needs THEN GOTO
870 NEXTI
880 GOSUB1000:REM SOUND
890 NEXTT
900 INPUT"WHAT WAS THE WORD ";G$
910 IF G$=X$ THEN PRINT"CORRECT":RETURN
920 PRINT"NO IT WAS ";X$
930 RETURN
999 REM**********SOUND**********
1000 FOR S=1TO1000:NEXT S
1010 FOR R=1 TO LEN(MC$(I))
1020 LET D$=MID$(MC$(I),R,1):PRINTD$;:REM SPECTRUM - LET
     D$=M$(R TO R):PRINT D$;
1030 IF D$="."THEN SOUND 1,-10,150,1:REM SPECTRUM - IF D$="."
     THEN BEEP .1,30
1040 IF D$="-"THEN SOUND 1,-10,150,10:REM SPECTRUM - IF D$="-"
     THEN BEEP 1,30
1050 FOR S=1TO1000:NEXT S
1060 *FX21,5
1065 REM ONLY BBC USERS SHOULD TYPE LINE 1060
1070 NEXT R
1080 PRINT
1090 FOR S=1TO1000;NEXT S
1100 RETURN
```

MORE VIC MODIFICATIONS

```
1030 IF D$="."THEN POKE36878,10:POKE36876,240:FORI=1TO100:
     NEXTI
1035 POKE36878,0:POKE36876,0
1040 IF D$="-"THEN POKE36878,10:POKE36876,240:FORI=ITO400:
     NEXTI
1045 POKE36878,0:POKE36876,0
```

PROJECT 4.
Spectrum owners will need to modify line 100 and to use single letter variables throughout.

```
  10 GOSUB100
  20 GOSUB200
  30 GOSUB300
  40 END
 100 DIM CO$(18),E(18),CU$(18):REM SPECTRUM DIM C$(18):
     DIME(18):DIMC$(18)
 105 @%=&20209:REM BBC USERS ONLY
 110 FORI=1TO18
 120 READ CO$(I),E(I),CU$(I)
 130 NEXT I
 140 RETURN
 200 PRINT"THIS IS A LIST OF COUNTRIES I HAVE DATA FOR :"
 210 FORI=1TO18:PRINTCO$(I):NEXT
 220 INPUT"WHICH COUNTRY DO YOU WANT ";W$
 230 FORI=1TO18:IF W$=CO$(I) THEN E=E(I):C$=CU$(I):GOTO260
 240 NEXTI
 250 PRINTW$;" IS NOT IN MY LIST !":GOTO220
 260 RETURN
 300 PRINT
 310 PRINT"DO YOU WANT - "
 320 PRINT"A - ";C$;" TO POUNDS ?"
 330 PRINT"B - POUNDS TO ";C$;" ?"
 340 INPUT"WHICH LETTER ";Q$
 350 IF Q$="A"THEN GOTO400
 360 IF Q$="B"THEN GOTO500
 370 GOTO340
 400 PRINT
 410 PRINT"HOW MANY ";C$;" ";:INPUTA$
 420 PRINTA$;" ";C$;" = ";VAL(A$)/E;" POUNDS"
 430 RETURN
 500 PRINT
 510 INPUT"HOW MANY POUNDS ";B$
 520 PRINTB$;" POUNDS = ";VAL(B$)*E;" ";C$
 530 RETURN
1000 DATA "AUSTRIA",26.05,"SCHILLINGS"
1010 DATA "BELGIUM",78.75,"FRANCS"
1020 DATA "CANADA",1.84,"DOLLARS"
```

```
1030 DATA "DENMARK",13.70,"KRONER"
1040 DATA "FRANCE",11.45,"FRANCS"
1050 DATA "W GERMANY",3.72,"MARKS"
1060 DATA "GREECE",148.00,"DRACHMAE"
1070 DATA "HOLLAND",4.21,"GUILDERS"
1080 DATA "IRELAND",1.22,"PUNT"
1090 DATA "ITALY",2310.00,"LIRE"
1100 DATA "MALTA",0.61,"POUNDS"
1110 DATA "NORWAY",10.78,"KRONER"
1120 DATA "PORTUGAL",186.00,"ESCUDOS"
1130 DATA "SPAIN",209.00,"PESETAS"
1140 DATA "SWEDEN",11.12,"KRONOR"
1150 DATA "SWITZERLAND",3.07,"FRANCS"
1160 DATA "USA",1.44,"DOLLARS"
1170 DATA "YUGOSLAVIA",180.00,"DINARS"
```

PROJECT 5.

```
 10 GOSUB100:REM INITIALISE
 20 GOSUB500:REM MAIN PROGRAM
 30 END
 99 REM********INITIALISE********
100 DIMS$(13),D$(13),R(13),W$(4),X$(4),Y$(4),Z$(4)
105 REM SPECTRUM USERS AMEND 100
110 FORI=1TO13:READS$(I),D$(I):PRINTI," ";D$(I);" ";S$(I):NEXT
120 FORI=1TO4:READW$(I),X$(I),Y$(I),Z$(I):NEXT
130 PRINT
140 RETURN
150 DATA"CAPRICORN","DEC21-JAN19"
160 DATA"AQUARIUS","JAN20-FEB18"
170 DATA"PISCES","FEB19-MAR20"
180 DATA"ARIES","MAR21-APR20"
190 DATA"TAURUS","APR21-MAY20"
200 DATA"GEMINI","MAY21-JUN20"
210 DATA"CANCER","JUN21-JUL21"
220 DATA"LEO","JUL22-AUG21"
230 DATA"VIRGO","AUG22-SEP22"
240 DATA"LIBRA","SEP23-OCT22"
250 DATA"SCORPIO","OCT23-NOV22"
260 DATA"SAGITTARIUS","NOV23-DEC20"
270 DATA"AQUARIUS","JAN20-FEB18
280 DATA"MEAN","MILLIONAIRE","RUN OVER","HELP"
290 DATA"PLEASANT","POPSTAR","LISTEN TO","SUCCESS"
300 DATA"HAPPY","PRINCESS","KISS","LOVE"
310 DATA"CLEVER","COMPUTER EXPERT","BUG","FAME"
499 REM******MAIN PROGRAM*********
500 INPUT"WHICH NUMBER IS YOUR STAR SIGN ";N
510 IF N<1ORN>13THEN GOTO500
520 PRINT
```

```
530 PRINT"FORECAST FOR ";S$(N)
540 REM****SELECT 4 RANDOM NUMBERS****
550 LET A=RND(4):REM VIC LET A=INT(RND(1)*4)+1:SPECTRUM LET
    A=INT(RND*4)+1
560 LET B=RND(4):REM VIC LET B=INT(RND(1)*4)+1:SPECTRUM LET
    B=INT(RND*4)+1
570 LET C=RND(4):REM VIC LET C=INT(RND(1)*4)+1:SPECTRUM LET
    C=INT(RND*4)+1
580 LET D=RND(4):REM VIC LET D=INT(RND(1)*4)+1:SPECTRUM LET
    D=INT(RND*4)+1
590 PRINT
600 PRINT"YOU ARE A VERY ";W$(A);" PERSON"
610 PRINT"TODAY YOU WILL MEET A ";X$(B)
620 PRINT"WHO WILL ";Y$(C);" YOU"
630 PRINTZ$(D);" IS ON THE WAY"
640 RETURN
```

PROJECT 6.

Spectrum owners need to make the same changes as before

```
  5 REM IAN THORNHILL
 10 INPUT"HIGHEST NUMBER ";HN
 20 LET P=RND(HN):REM VIC LET P=INT(RND(1)*HN)
 30 PRINT"I'VE CHOSEN A NUMBER BETWEEN 1 AND ";HN
 40 INPUT"WHAT'S YOUR NUMBER ";Q
 50 PRINT"DO YOU WANT TO KNOW WHETHER ";Q
 60 PRINT"IS HIGHER OR LOWER THAN MY NUMBER?"
 70 INPUT"H OR N ";HL$
 80 IF HL$="H"AND P>Q THEN PRINT"YES IT IS HIGHER THAN ";Q:
    GOTO120
 90 IF HL$="H"AND P<Q OR P=Q THEN PRINT"NO IT IS NOT HIGHER
    THAN ";Q:GOTO120
100 IF HL$="L"AND P<Q THEN PRINT"YES IT IS LOWER THAN ";Q:
    GOTO120
110 IF HL$="L"AND P>Q OR P=Q THEN PRINT"NO IT IS NOT LOWER THAN ";Q
120 INPUT"DO YOU WANT TO GUESS MY NUMBER (Y/N) ";YN$
130 IF YN$="Y" THEN PRINT"WHAT IS YOUR GUESS":GOTO150
140 IF YN$="N" THEN GOTO40
150 INPUT G
160 IF G=P THEN PRINT"YES";G;"WAS THE CORRECT ANSWER":GOTO
    170
170 IF G<>P THEN PRINT"NO THAT IS WRONG":GOTO40
180 INPUT"ANOTHER GAME(Y/N) ";AG$
190 IF AG$="Y"OR AG$="YES" THEN GOTO10
200 END
```

PROJECT 7.

Very few people manage to spot the 'quick way' - that is if a number is exactly divisible

by three numbers then it must be a multiple of those three numers - so the answer was 7 x 8 x 9 = 504.

However, the computer is not capable of thinking in an intelligent way so we have to adopt another approach.

Try this -

```
10 FORI=100TO999
20 IF(I+7)/7<>INT((I+7)/7)THEN60
30 IF(I+8)/8<>INT((I+8)/8)THEN60
40 IF(I+9)/9<>INT((I+9)/9)THEN60
50 PRINTI
60 NEXTI
```

PROJECT 8.

The algorithm could be

1) Set OLD to 1111
2) Multiply OLD by (4 - new)
3) Reverse (OLD - REV)
4) Is REV equal to NEW?
5) If not, increment OLD by 1 (as far as 2500) and repeat
6) If yes, print OLD and NEW
7) Continue to 2500

```
 10 OLD$="1111"
 20 NE$=RIGHT$(STR$(4*(VAL(OLD$))),4)
 30 REM  CALCULATES 4 * VALUE OF OLD NO.
 40 FORI=4TO1STEP-1
 50 REV$=REV$+MID$(OLD$,I,1)
 60 REM  REVERSES OLD NO.
 70 NEXTI
 80 IFREV$=NE$THENPRINTOLD$,NE$
 90 REM PRINTS ANSWER IF NEW NO. = OLD NO. REVERSED
100 OLD$=RIGHT$(STR$(VAL(OLD$)+1),4)
110 REM  ADDS 1 TO OLD NO.
120 IFOLD$="2500"THEN160
130 NE$="":REV$=""
140 REM  READY TO START AGAIN
150 GOTO20
160 END
```

PROJECT 9.

This is one way of doing it:

```
10 REM CHRIS SKELLAM
90 DIMR$(10,100)
```

```
 95 DIMC(100)
 97 TI$="000000"
100 FORN=31623 TO 99999
102 M=INT(N/1000):R=M*M
104 L=N-1000*INT(N/1000):P=L*L
106 Q=M*L*2:U=P-1000*INT(P/1000)
108 R$=STR$(U):T=INT(P/1000)+Q-1000*INT(Q/1000)
110 T$=STR$(T-1000*INT(T/1000))
112 H=R+INT(Q/1000)+INT(T/1000):H$=STR$(H)
114 IFLEN(R$)<3THEN:R$="00"+RIGHT$(R$,1)
116 IFLEN(R$)<4THEN:R$="0"+RIGHT$(R$,2)
117 R$=RIGHT$(R$,3)
118 IFLEN(T$)<3THEN:T$="00"+RIGHT$(T$,1)
120 IFLEN(T$)<4THEN:T$="0"+RIGHT$(T$,2)
122 T$=RIGHT$(T$,3)
125 S$=H$+T$+R$
126 PRINTS$
130 FORJ=0TO9
140 N(J)=0
150 FORP=2TO LEN(S$)
160 IF VAL(MID$(S$,P,1))=J THEN:N(J)=N(J)+1
170 NEXTP
180 IFN(J)>M(J)THEN:M(J)=N(J):C(J)=1:R$(J,C(J))=S$:GOTO200
185 IFN(J)=M(J)THEN:C(J)=C(J)+1:R$(J,C(J))=S$
190 NEXTJ
200 NEXTN
210 FORJ=0TO9
220 PRINT "LARGEST NUMBER OF";J;"'S IS ";M(J)
222 PRINT
225 FORI=1TOC(J)
226 PRINT R$(JI),INT(SQR(VAL(R$(J,I)))+5)
230 NEXTI
233 PRINT:PRINT:PRINT
235 NEXTJ
240 PRINT:PRINT TI$
```

PROJECT 10.

This VIC-20 program would take about 8 hrs

```
10 LET TI$="000000"
20 FOR I=1 TO 1000000
30 PRINT I
40 NEXT I
50 PRINT TI$
60 END
```

By leaving out the variable I in line 40 an increase in speed could be achieved.

```
10 TI$="000000"
20 FORI=1TO1000000:NEXT
30 PRINTTI$
40 END
```

So this program reaches 1000000 in 18 minutes and 11 seconds.
The increase in speed is due to the fact that the CPU doesn't have to keep looking up the values of I before it increments it.

You can see how much difference this makes on a shorter time scale by running this program.

```
20 TI$="000000"
30 FORI=1TO1000:NEXTI
40 PRINT"  WITH I :";TI;"JIFFIES"
50 TI$="000000"
60 FOR I = 1 TO 1000:NEXT
70 PRINT"WITHOUT I :";TI;"JIFFIES"
80 END
```

PROJECT 11.

Here is one possible program:

```
 10 REM CHRIS SKELLAM
 20 LET C=6
 30 LET C=C+5
 40 DIM B(6)
 50 LET D=1
 60 LET T=C
 70 FOR A=1 TO 6
 80 LET M=T-1
 90 LET F=M/5
100 IF <>7 INT F THEN GOTO 30
110 LET T=T-F-1
120 LET B(D)=F
130 LET D=D+1
140 NEXT A
150 PRINT"THE LOWEST POSSIBLE NO. OF COCONUTS IS ";C
155 PRINT
160 FOR A=1 TO 5
170 PRINT"SAILOR ";A;" GOT ";B(A)+B(6);" COCONUTS.
180 NEXT A
190 PRINT
200 PRINT "(THE MONKEY GOT 6.)"
```

The correct answer was of course 15,621.

CHAPTER 10

The Future

Improvements in the design and manufacture of microcomputers have occurred remarkably quickly and it seems likely that this phenomenon will continue at an ever-increasing rate. To illustrate this almost exponential growth, take the first generally available micro-computer, the Commodore PET. In 1979 this cost £600 and had 16K ROM and 8K RAM. Now, just five years later, there is already a 64K ROM chip in production and a 1000000K RAM chip under development in Japan. The home-produced Sinclair QL (for Quantum Leap!) costs £399, has 32K ROM and 128K RAM as standard. The ROM is expandable through cartridges and RAM can be expanded to 640K!

In addition, the QL has inbuilt software (word-processor, database and spreadsheet) and it seems likely that integral software will become a feature of the newer computers.

But some of the more recent developments in software have been in so-called "expert systems". These programs are really huge databases of knowledge compiled by experts in areas such as medicine, law, engineering, etc. These systems are easy to use and can be interrogated and amended by using simple non-computing terms. For example, MYCIN, an expert system in medicine, allows a doctor to input a patient's symptoms and then read the computer's suggested diagnosis (or rather that of the expert who helped program the computer!). In addition, information about new drugs etc., can easily be incorporated into the system.

Other expert systems include:

R1 - a decision-making system
CYRUS - a question and answer system
PROSPECTOR - a geological problem solver
EURISKO - a system to solve mathematical problems

To make the design of expert systems easier, a new computer language has been used in some cases. This language, called PROLOG (PROgramming in LOGic), at the moment is only available for mini- and main-frame computers, but as the memory capacity of microcomputers increases, micro-versions of PROLOG will become available.

Some experts think PROLOG is so good that they suggest it ought to be the new programming language for *all* computers. Perhaps this is one of the reasons why

the Japanese have chosen PROLOG as the operating language for their "Fifth Generation" computers. In the race to produce one of these I.K.B.S.'s (Intelligent Knowledge Based Systems), as the fifth generation computers are sometimes known, the Japanese lead the world in the technology and hope to have the first IKBS computer "up and running" in 1986. But since the British lead the world in software design, the Japanese will rely on the U.K. to supply software.

Developments like these, in large computer design, will eventually be incorporated into micros; one of the newer design changes already available is in information input. The light pen and stylus are available from most computer shops and these are particularly useful for handicapped people who have difficulty in typing. Another device to make data entry easier is the Concept keyboard - a much enlarged keyboard with touch sensitive squares which can be programmed for whole word entry as well as single letters. Another micro now available has a touch sensitive screen so that the user can communicate "directly" with the program.

An offshoot from computer aided design is the "mouse". This device, which looks like the animal it gets its name from (but with wheels!), behaves rather like an intelligent joystick. It plugs into the micro and by moving the mouse around on a special drawing board, the computer can plot the exact position of the mouse on the screen. It is therefore possible to trace over pictures laid on the drawing board and see the tracing instantly on the computer's VDU. The finished picture can then be stored, coloured, rotated etc., by pushing buttons on the mouse.

Work is also progressing rapidly on voice input and a specially built micro can now open doors, put on the light, change channels on the television set etc., on "hearing" the command! The same computer can also be programmed to respond to just one voice as well as translate from one language to another! All this of course is intended to make the computer easier to use.

Similarly, computer languages like LOGO and FORTH try to make programming the computer much easier. FORTH however allows the user to invent his or her own statements for programming and this overcomes the difficulty of incompatability of BASIC dialects.

You will have noticed that most of the programs in this book have had to be written in three versions to match the variations of BASIC in the VIC 20, the Spectrum and the BBC machines. Now the BBC itself has gone part way to solving the problem by broadcasting software in a radio programme called "Chip Shop". The programs are written in a form of BASIC called BASICODE. This program can then be changed by a translation program into a machine-specific program.

The Basicode system, first developed in Holland, is in use in Belgium, West Germany, Australia, New Zealand, U.S.A. and Canada. So it means that we can now exchange programs with penfriends even if they have a different machine.

The BASICODE translation cassette is available in the U.K. from

BBC, P.O. Box 7, LONDON. W3 6XJ

Now for a brief look at what is available for your computer now or in the near future.

The BBC machine and the Future

One of the problems of the "New Technology" is that many ideas are out of date almost before they are fully available. The designers of the BBC computer have tried to ensure that any future developments in microcomputer technology can be easily incorporated into the present machine. This is due to a system called "The Tube" which is both a connector and a piece of software. The tube will allow the BBC to communicate not only with a second 6502 processor but also a Z80, the main processor of Sinclair computers. In addition, much faster, more complex processors are planned which will allow the BBC machine to do the "chores" while the second processor gets on with the main program.

However, as a reader of this book you will probably also be interested in the "peripherals" which are available now for your BBC machine:

Printers and Disk Drives

Adding a printer to your computer equipment enables you not only to produce listings of your programs but, by using a program called a word processor, you write letters or essays which look professionally produced. Indeed the original text for the words you are now reading were written using a word processor. With the right printer you can also to produce pictures like the one below.

The storage of your programs is as important as the programs themselves. At the moment you will probably be using a tape recorder to do this. Although this is a fairly efficient method, it is very slow especially when compared with disk storage. A program which takes four minutes to load from tape will load in almost as many seconds from disk! For those who have seen too much of the message "Block?" or "Data?", far fewer errors are encountered when reading a program from a disk!

Each disk can normally hold up to 100K of information. This means roughly ten to twenty fairly long programs, though double sided and double density disks are available at extra cost.

Teletext adaptor

This unit, together with a chip to fit inside your computer and a good aerial, will turn your computer T.V. or monitor into a CEEFAX and ORACLE receiver. As such, you will be able not only to call up pages of interest such as Sport, News, Pops, Jokes etc., but also "capture" computer programs for your own use. Viewers' own programs can also be broadcast and for these there is usually a small prize!

Many of these programs have been written by Telsoft, a team of programmers at Brighton Polytechnic who are interested in education and computers. Some of the software is "interactive" in that the program "interrogates" various CEEFAX pages in the same way as a database program works. So, for example, a certain program may require you to draw up a menu for a meal for six people. You will have to make this meal as nutritious as possible but on a budget of £10. By referring to the Food Prices pages of CEEFAX you can get up to the minute information and then decide which products to buy. Having decided on your menu, the computer will then analyse your choices and display its comments.

Prestel adaptor

This add-on unit is similar to the Teletext unit except that it connects via a telephone link to the G.P.O.'s computers. If you take the example given of the food program then the big difference between Teletext and Prestel is that you could actually order the food from a supplier who is connected to Prestel. You can use this system to shop from the comfort of your armchair! Unfortunately, it does cost more than Teletext as the phone link has to be paid for and usually there is a charge for each page of information viewed!! However, there are many thousands of pages available compared with only hundreds for Teletext.

The 1MHz Bus

If you look under the keyboard of your BBC micro you will find several connectors labelled "disk drive", "printer" etc. The 1MHz bus is the connector by which the BBC computer can be linked up to special equipment. This includes laboratory machinery, analogue to digital converters (the picture on previous page was converted from a photograph to a digital picture that the computer could understand) and eprom programmers.

An eprom is an Erasable Programmable Read Only Memory chip. It has the advantage that a program can be "burned" into it but "washed" out later if you want. Many suppliers are now offering computer programs on these chips. These programs have the advantage of being available immediately for use. Imagine a space invader program burned into an eprom - all you would have to do to play it would be to type *INVADER and it would immediately run! No loading at all!

The BBC can support sixteen of these chips altogether in a special part of the circuitry call the "Sideways ROMS". One of the problems is that this circuitry is inside the case so if you wanted to change chips, you need to take off the cover. However, several firms are now manufacturing ROM boards which sit outside the machine but have a cable connected inside the machine. Once fitted you can change chips without lifting the lid.

Spectrum

Before buying any further equipment for your Spectrum, be sure you know which of the three different models you have. Model 1 differs in important respects from Models 2 and 3, but Models 2 and 3 are very similar. Check that your intended purchase will work with your mode

To find out which model you have, look at the edge connector at the back of your Spectrum. Near the power input is a rectangular slot through which you can see part of the computer's base. Now, if the gold coloured strips are the same width as the white insulating strips, then your Spectrum is a Model 1. But if the gold strip is twice as wide as the white, then yours is either a Model 2 OR 3. To find out which, type PRINT IN 57342. A Model 2 will reply with 255. A Model 3 will reply with 191.

The next problem is to check the memory size - many 16K Spectrums have been upgraded internally. Check the RAM size by typing PRINT PEEK 23733. A 16K RAM will reply with 128 while a 48K RAM will give 255. If your machine has a 16K RAM then it is quite inexpensive and well worth upgrading it to 48K.

Printers and disk drives

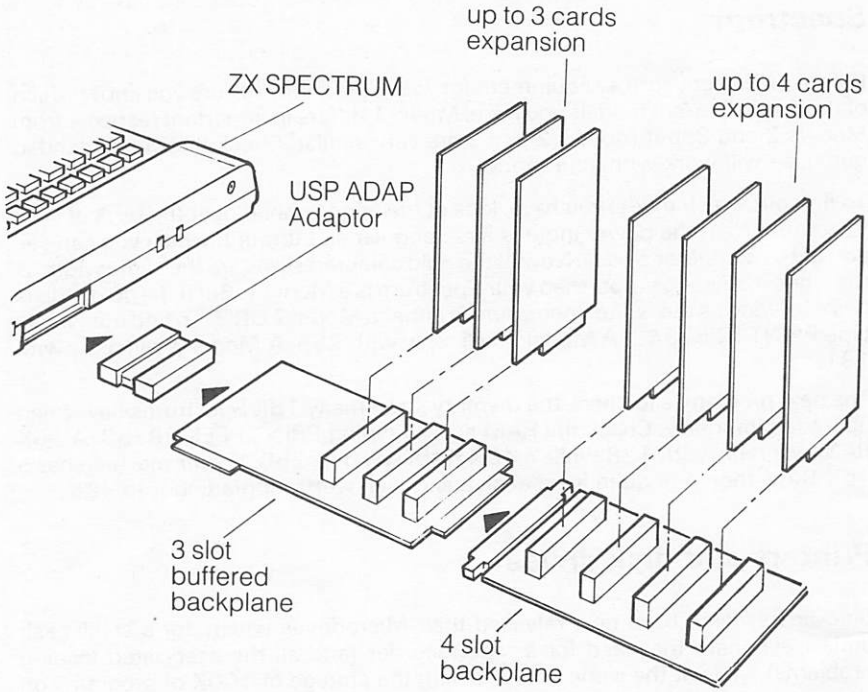
Sinclair Research have now released their Microdrives which, for a small cash outlay, eliminate the need for a tape recorder (and all the associated loading problems) while at the same time allowing the storage of 100K of programs on one cartridge. In addition the loading, saving and accessing times are very much quicker than those for tape operation.

The Sinclair printer is a very inexpensive device which will enable you to make hard copy listings of your programs and draw simple graphics pictures. As the printer uses specially made aluminium coated thermal paper, it is a little restricting for the more adventurous. However, an RS-232 interface will allow the Spectrum to be connected to most standard printers. With one of these, word processing then becomes a possibility.

Other "add-ons"

A modem is now on sale which will enable you to connect your Spectrum to the teletext system, and, via a Micronet modem, connection to a subset of Prestel is also possible. These items will greatly enhance the power of your Spectrum.

An exciting prospect for the Spectrum owner is the "motherboard" idea. This is a circuit which is connected to the Spectrum's edge connector. By doing this, a wide range of other circuit boards can be slotted into the motherboard. An arrangement would look something like this:



(By kind permission of U-Microcomputers)

There is also the possibility of using a language other than BASIC on your Spectrum. A LOGO compiler is already available from a few sources and, as already mentioned, Micro-PROLOG is very near. Those of you who are interested in Artificial Intelligence may like to get a copy of Micro-PROLOG, which is favoured by many A.I. researchers. FORTH is an expandable language in which, as already stated, the user creates his or her own keyboards, though a small "nucleus" of commands is present to help get you started.

VIC 20

The unexpanded VIC 20 has a small memory capacity - just 3K. Consequently, the first expansion to your set-up is to increase the memory size so that longer programs can be run. Many companies make RAM expansion packs, but some are designed for use on a motherboard whilst others are connected directly to the computer. Although the motherboard-type expansions cost more, they are the better buy if you intend adding any more circuitry to perhaps increase the ROM (like Toolkit chips etc.). We advise you to purchase switchable RAM expansions which, though more expensive, are more versatile.

Printers and disk drives

There is a wide variety of printers available for the VIC 20. However the VIC connects to a printer through its IEEE-48 port. This is a type of connector which is now not much used except in the engineering field. Consequently, most of these non-Commodore printers will either need modification or an extra unit to connect between them and the VIC. Our advice is, therefore, buy a Commodore printer.

The VIC 20 uses the Commodore tape recorder which although extremely reliable is very slow. The VIC disk drive which although expensive is a must for the really serious user. The storage capability is 174K bytes of data in program form. Once you get used to the speed of a disk drive (250 characters per second transfer rate!) it becomes almost impossible to go back to tape. A further disadvantage of tape is that if you want to load, say, the fifth program, you have to wade through the other four which come before it on the tape, since unlike the BBC machine, the VIC gives no indication of what is entering the machine!

Other "add-ons"

Like the other machines dealt with in this book, the VIC 20 can be enhanced by a very wide range of "peripherals". There is a Teletext decoder to be available soon, and a Micronet (Prestel subset) modem is available now. There are light pens, graph plotters, joysticks, voice synthesisers etc.,etc., But, as the VIC 20 is no longer manufactured, we hesitate to recommend any of them. Many VIC 20 owners will doubtless upgrade to the more recently introduced Commodore 64 range of computers.

You can keep up with developments in the whole Commodore range by contacting the Independent Commodore Products User Group at:

ICPUG,
30, Brancaster Road,
Newbury Park,
ILFORD,
Essex.
IG2 7EP

Conclusion

If you have read through all of this book, you have reached the end of a long journey! Thank you for staying the course. If you have understood and enjoyed what you have read, then we are convinced that you are ready to enter the exciting world of microcomputing with confidence. At times you must have felt like giving up. This feeling of helplessness affects all computer users at one time or another. Remember the successful completion of a program is always a sufficient reward for hours of patient effort.

Whether you are thinking of going on to Further Education to study Computing, setting up a software business or just "playing" with your computer, you have our very best wishes!!!

INDEX

For each keyword, **bold** indicates an explanation, normal type refers to the use of the keyword in a program.

ATTR (Spectrum) **69**

BRIGHT (Spectrum) 62
BORDER (Spectrum) **62, 63**

CAPS SHIFT (Spectrum) *10*, 68
CHR$ **68,** 80
CIRCLE (Spectrum) *60*, 61, 64, 66
CLOSE # 105
CLS *64*, 69,
COPY (B.B.C) **10**

DATA **20**,21,23,34
DEF FN **36,**
DIM (BBC + VIC) **32**,35,113,134
(Spectrum) **35**
DRAW (Spectrum) **59**,60,61,65
DRAW (BBC) **70**,71,72

EDIT (Spectrum) **10**
END **9,** 11, 30
ENVELOPE (BBC) **95,**

GET **35**
GOSUB **23**,24,25,54
GOTO **18**,22,23,28

FOR **13**,20,22
FLASH (Spectrum) **62**

IF **24**,29
INKEY$ (Spectrum) **35**,66
INPUT **11**,12,13
INPUT$ **16**,17,18
INPUT # 105
INT (BBC + VIC) 25,28,132
INT (Spectrum) 25,28,132
INVERSE (Spectrum) **63**

LEFT$ 30
LEN **31**
LET **9**,13,20,25
LIST **10**,11

MID$ **30**,31,32,122
MODE (BBC) **70**,71,72,75
MOVE (BBC) **70**,71,73,75

NEXT **13**,20,22
NEW **9**,

ON **24**
OPEN 105
OVER (Spectrum) **62**

PAPER (Spectrum) **62**,64
PAUSE (Spectrum) **66**,69,70
PEEK **6**,3,7
PLOT (Spectrum) **59**,60,66
PLOT (BBC) **74**,75,76
POKE **6**,64,68
PRINT **9**,3,4,11
PRINT with $ **15**
PRINT # 105
PRINT AT (Spectrum) **59**,61,63,64

READ **20**,21,23,34
REM 116,123,128
RESTORE **21**
RETURN **23**
RIGHT$ **30**,133
RND (BBC) **28**
RND (Spectrum) **28**,25
RND (VIC) **28**

SCREEN$ (Spectrum) **69**
STEP 60,75,92
SOUND (BBC) **92**
SQR **42**

TAB **18**
THEN *29*,44,45

Join the Winning Team!

Sigma Press have an enviable reputation for producing a top-class range of computer books. Soon there'll be over a hundred, spanning all aspects of computing from BASIC to Artificial Intelligence. We concentrate on the upper-end of the market, so that even our books on very popular micros tend to emphasise techniques and applications other than collections of games.

Perhaps you could write a book for us. If so, write to the publisher, Graham Beech at the address below for full details - including a helpful advice sheet on how to submit a synopsis of your book in the best possible way.

We are interested in **all** aspects of computing - after all, we ourselves are computer enthusiasts! Dealing with Sigma is dealing with someone who understands **your** subject - someone who can advise you on what to emphasise in your book to make it a success. That's so much better than dealing with a multinational corporation!

Sigma work **with** technology, not against it. So, we welcome authors who submit manuscripts on word-processor disks from which we typeset automatically. This is just one of the methods we use to get your book on the market in the shortest possible time.

Marketing and distribution is, of course, of the highest calibre - through John Wiley & Sons Ltd., who are a well-known international publisher and with whom we have worked for several years with great success.

So, when you want your book to be a success, just write to:

Graham Beech
Sigma Press
5 Alton Road,
Wilmslow,
Cheshire.
SK9 5DY.

or, telephone: 0625 - 531035

THE YOUNG PERSON'S GUIDE TO COMPUTING

Many people - parents and children - get a computer, but don't know what to do with it. Once Space Invaders has been used for a couple of weeks, the novelty begins to fade. Just what can you do with a microcomputer?

Well, here's the answer. You learn about all aspects of how computers work and what they can do to make life easier. You learn how to solve problems, to play serious and challenging games, produce exciting graphics.

Perhaps even more important, you'll find that this book covers those very computing topics that children from 9 to 16 have to learn at school.

The book covers most major aspects of Computer Studies up to CSE and 'O' level, including all important aspects of BASIC. It also covers many other important computing topics such as algorithm design, flowcharting, program debugging, and aspects of machine code programming. In addition to these software aspects, the design of microcomputers is illustrated by reference to currently available chips.

Microsoft BASIC is presented as a unifying standard language. Dialects are discussed for the following machines that are popular in UK schools: BBC Model A and B,Research Machines 380Z, Commodore 64, VIC-20 and Sinclair Spectrum.

You can learn almost everything you ever need to know about computers from our books.

Write for a catalogue:

Sigma Technical Press
5 Alton Road
Wilmslow
Cheshire
SK9 5DY

£6.50

ISBN 0 905104 63 3